QUANTIFICATION AND FORMALIZATION OF SECURITY

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Michael Ryan Clarkson

February 2010

QUANTIFICATION AND FORMALIZATION OF SECURITY

Michael Ryan Clarkson, Ph.D.

Cornell University 2010

Computer security policies often are stated informally in terms of confidentiality, integrity, and availability of information and resources; these policies can be qualitative or quantitative. To formally quantify confidentiality and integrity, a new model of quantitative information flow is proposed in which information flow is quantified as the change in the accuracy of an observer's beliefs. This new model resolves anomalies present in previous quantitative information-flow models, which are based on change in uncertainty. And the new model is sufficiently general that it can be instantiated to measure either accuracy or uncertainty. To formalize security policies in general, a generalization of the theory of trace properties (originally developed for program verification) is proposed. Security policies are modeled as hyperproperties, which are sets of trace properties. Although important security policies, such as secure information flow, cannot be expressed as trace properties, they can be expressed as hyperproperties. Safety and liveness are generalized from trace properties to hyperproperties, and every hyperproperty is shown to be the intersection of a safety hyperproperty and a liveness hyperproperty. Verification, refinement, and topology of hyperproperties are also addressed. Hyperproperties for system representations beyond trace sets are investigated.

BIOGRAPHICAL SKETCH

Michael Clarkson received the Indiana Academic Honors Diploma from the Indiana Academy for Science, Mathematics, and Humanities in 1995. He received the B.S. with honors in Applied Science (Systems Analysis) and the B.M. in Music Performance (Piano) from Miami University in 1999, both *summa cum laude*. The Systems Analysis curriculum was a combination of studies in computer systems, software engineering, and operations research. As part of an experimental branch of that curriculum, he studied formal methods of software development. He received the M.S. in Computer Science from Cornell University in 2004. As part of his doctoral studies at Cornell, he completed a graduate minor in music including studies in organ, conducting, and voice.

Remote as we are from perfect knowledge,

we deem it less blameworthy to say too little,

rather than nothing at all.

—St. Jerome

ACKNOWLEDGEMENTS

I thank all the institutions that directly supported me through fellowships: a Cornell University Fellowship (2000), a National Science Foundation Graduate Research Fellowship (2001), and an Intel Foundation Fellowship (2007). The work reported in this dissertation was also supported in part by the Department of the Navy, Office of Naval Research, ONR Grant N00014-01-1-0968; Air Force Office of Scientific Research, Air Force Materiel Command, USAF, grant F9550-06-0019; National Science Foundation grants 0208642, 0133302, 0430161; AF-TRUST (Air Force Team for Research in Ubiquitous Secure Technology for GIG/NCES), which receives support from the Air Force Office of Scientific Research (FA9550-06-1-0244), the National Science Foundation (CCF-0424422), BT, Cisco, ESCHER, HP, IBM, iCAST, Intel, Microsoft, ORNL, Pirelli, Qualcomm, Sun, Symantec, Telecom Italia, and United Technologies; a grant from Intel; and a gift from Microsoft.

Research is neither an art nor a science, but a craft that requires special skill and careful attention to detail. As in the Middle Ages, this craft is taught via apprenticeship. I had the good fortune to apprentice myself to two outstanding masters of the craft, Andrew Myers and Fred B. Schneider. Both were essential to my training, and I profited from working with both—despite Matthew 6:24, "No man can serve two masters." This dissertation would not exist if not for their ample ideas, critiques, and advice. The skills and habits that I learned from these researchers are inestimable and practically innumerable. But I thank Andrew most of all for the habit of steadfast persistence and curiosity in the pursuit of research. And I thank Fred most of all for the habit of lucid writing. Perfection of these habits is something I will pursue throughout my life, with their voices guiding me.

Three other Cornell faculty especially deserve mention. Graeme Bailey and Dexter Kozen helped me to solve mathematical problems. And Kavita Bala gave me surprisingly useful advice in my last semester.

The faculty of Miami University introduced me to scholarship. Gerald Miller first predicted that I would take the path toward a Ph.D., and Ann Sobel made certain that I did. Ann was my first master, and I thank her for helping me find my way to Cornell. Alton Sanders and Michael Zmuda tutored me in subjects in which Miami did not offer classes. James Kiper was an influential role model for teaching. Donald Byrkett gave me my first teaching assistantship, in my first semester at Miami. Douglas Troy gave me my first teaching appointment, in my last semester at Miami, and he gave me my first job offer (albeit tongue-in-cheek) for an assistant professorship. Richard Nault predicted where this path would end; only time will tell.

To my fellow apprentices—Steve Chong, Matthew Fluet, Nate Nystrom, Kevin O'Neill, and Tom Roeder: thank you most of all for your friendship.

To the staff who supported me not just as a student, but as a person—Stephanie Meik and Becky Stewart: thank you for allowing me to distract you from your daily work, and for listening. Your nurture and counsel have made my life better.

Steve Chong once said to me that graduate school was supposed to be a process of losing hobbies, but somehow I kept gaining them. For the recreation of gaming, I thank Joseph Ford, Karen Downey, William Hurley, and Rob Knepper. For the satisfaction of food and drink, I thank the Cornell Hotel School wine and cooking classes, Standing Stone Vineyards, Gimme! coffee, and the department espresso machine. For the inspiration of music, I thank my music teachers at Cornell—Thom Baker, Chris Kim, James Patrick Miller, Timothy Olsen, Annette Richards, and David Yearsley—and my musician–friends—Heidi Miller, Catherine Oertel, Emily Sensenbach, Bob Whalen, and the members of the Cornell Chorale.

I cannot adequately thank my parents, Dennis and Rhonda, but I will try by continuing to live by their good example.

Finally, medieval apprentices were not allowed to wed, but I married my high school sweetheart, Rachel. I cannot express my gratitude or debt to her, so I shall simply say: I love you, always and forever.

Michael Clarkson

Whit Sunday 2009

Ithaca, New York

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Computer *security policies* express what computer systems may and may not do. For example, a security policy might stipulate that a system may not allow a user to read information that belongs to other users, or that a system may process transactions only if they are recorded in an audit log, or that a system may not delay too long in making a resource accessible to a user.[1]

This dissertation addresses mathematical foundations for security policies, in two ways. First, metrics are developed for quantifying how much secret information a computer system can leak, and for quantifying the amount of trusted information within a computer system that becomes contaminated. Second, a taxonomy is proposed for formal, mathematical expression and classification of security policies. These contributions are best understood in the context of a select history of computer security policies.

1.1 Historical Background

Security policies have long been formulated in terms of a tripartite taxonomy: confidentiality, integrity, and availability. Henceforth, this is called the *CIA taxonomy*. There is no agreement on how to define each element of this taxonomy—as evidenced by table 1.1, which summarizes the evolution of the CIA taxonomy in academic literature, standards, and textbooks.[2] Perhaps the most widely ac-

[1]Security policies might also express what human users of computer systems may or may not do—for example, that users may not remove machines from a building. This dissertation focuses on computers, not humans; Sterne [111] discusses the relationship between these two kinds of policies.

[2]Nor is there agreement on what abstract noun to associate with the elements of this taxonomy. Various authors use the terms “aspects” [16, 47], “categories of protection” [31], “characteristics” [97], “goals” [26, 97], “needs” [72], “properties” [58], “qualities” [97], and “requirements” [92].

Table 1.1: Definitions of the CIA taxonomy. Confidentiality, integrity, and availability are abbreviated C., I., and A.

Source	Year	Term	Definition
Voydock and Kent [121]	1983	N/A	Security violations can be divided into… unauthorized release of information, unauthorized modification of information, or unauthorized denial of resource use.
Clark and Wilson [26]	1987	N/A	System should prevent unauthorized disclosure or theft of information, …unauthorized modification of information, and…denial of service.
ISO 7498-2 [58]	1989	C.	Information is not made available or disclosed to unauthorized individuals, entities, or processes.
		I.	Data has not been altered or destroyed in an unauthorized manner.
		A.	Being accessible and useable upon demand by an authorized entity.
ITSEC [30]	1991	C.	Prevention of unauthorized disclosure of information.
		I.	Prevention of unauthorized modification of information.
		A.	Prevention of unauthorized withholding of information or resources.
NRC [92]	1991	C.	Controlling who gets to read information.
		I.	Assuring that information and programs are changed only in a specified and authorized manner.
		A.	Assuring that authorized users have continued access to information and resources.
Pfleeger [97]	1997	C.	The assets of a computing system are accessible only by authorized parties. The type of access is read-type access.
		I.	Assets can be modified only by authorized parties or only in authorized ways.
		A.	Assets are accessible to authorized parties.
Gollmann [47]	1999	C., I., A.	Same as ITSEC.
Lampson [72]	2000	Secrecy	Controlling who gets to read information.
		I.	Controlling how information changes or resources are used.
		A.	Providing prompt access to information and resources.
Bishop [16]	2003	C.	Concealment of information or resources.
		I.	Trustworthiness of data or resources…usually phrased in terms of preventing improper or unauthorized change.
		A.	The ability to use the information or resource desired.
Common Criteria [31]	2006	C.	Protection of assets from unauthorized disclosure.
		I.	Protection of assets from unauthorized modification.
		A.	Protection of assets from loss of use.

cepted, current definitions (if only because of adoption by North American and European governments) are those given by the Common Criteria [31, §1.4], an international standard for evaluation of computer system security:

- *Confidentiality* is the protection of assets from unauthorized disclosure.
- *Integrity* is the protection of assets from unauthorized modification.
- *Availability* is the protection of assets from loss of use.

The term "assets" is essentially undefined by the Common Criteria. From the other definitions in table 1.1, we surmise that assets include information and system resources.

These definitions of the CIA taxonomy raise the question of how to distinguish between unauthorized and authorized actions. *Authorization policies* have been developed to answer this question. In the vocabulary of authorization policies, a *subject* generalizes the notion of a user to include programs running on behalf of users. Likewise, *object* generalizes "information" and "resource," and *right* is used instead of "action." Every subject can also be treated as an object, so that subjects can have rights to other subjects. Authorization policies can be categorized as follows:

- *Access-control policies* regulate actions directly by specifying for each subject and object exactly what rights the subject has to the object. File-system permissions (e.g., in Unix or Microsoft Windows) embody a familiar example of an access-control policy, in which users may (or may not) read, write, and execute files. Access-control policies originated in the development of multiprogrammed systems for the purpose of preventing one user's program from harming another user's program or data [74].[3]

[3]Lampson [74] gives the canonical formalization of access-control policies as matrices in which rows represent subjects, columns represent objects, and entries are rights.

- *Information-flow policies* regulate actions indirectly by specifying, for each subject and object, whether information is allowed to flow between them. This specification is used to determine what actions are allowed. *Multilevel security*, formalized by Bell and LaPadula [13] and by Feiertag et al. [43], is a familiar example of an information-flow policy that is used to govern confidentiality: Each subject is associated with a *security level* comprising a hierarchical *clearance* (e.g., Top Secret, Secret, or Unclassified) and a non-hierarchical *category set* (e.g., {Atomic, NATO}). Information is permitted to flow from a subject S_1 to subject S_2 only if the clearance of S_1 is less than or equal to the clearance of S_2 and the category set of S_1 is a subset of the category set of S_2.[4] *Noninterference*, defined by Goguen and Meseguer [46], is another, important example of an information-flow policy. It stipulates commands executed on behalf of users holding high clearances have no effect on system behavior observed by users holding low clearances. This policy, or a variant of it, is enforced by many programming language-based mechanisms [104].

When used to govern confidentiality of information, access-control policies regulate the release of information in a system, whereas information-flow policies regulate both the release and propagation of information. Thus information-flow policies are stronger than access-control policies. For example, an information-flow policy might require that the information in file `f.txt` does not become known to any user other than `alice`. A Unix access-control policy on file `f.txt` might approximate the information-flow policy by stipulating that

[4]The first mathematical formalization of security-level comparison seems to be a result of Weissman [124]; a more general formalization in terms of lattices was given by Denning [36]. Differences between the Bell–LaPadula and Feiertag et al. models of multilevel security are discussed by Taylor [114]. Multilevel security, in addition to being an information-flow policy, is an example of a *mandatory access control* (MAC) policy. In contrast are *discretionary access control* (DAC) policies—for example, Unix file-system permissions.

only `alice` can execute a read operation on `f.txt`. But a Trojan horse[5] running with the permissions of `alice` would be allowed, according to the access-control policy, to copy `f.txt` to some public file from which anyone may read. The contents of `f.txt` would no longer be secret, violating the information-flow policy.

Malicious programs such as a Trojan horse might exploit *channels*, or communication paths, other than the file system to violate information-flow policies. Lampson introduces the notion of a *covert* channel, which is a channel "not intended for information transfer at all" [73]—for example, filesystem locks, system load, power consumption, or execution time.[6] The Department of Defense later defined a covert channel somewhat differently in its Trusted Computer System Evaluation Criteria—also known as the "Orange Book" because of its cover—as "any communication channel that can be exploited by a process to transfer information in a manner that violates the system's security policy" [37]. The TCSEC categorizes covert channels into *storage* and *timing channels*. Storage channels involve reading and writing of storage locations, whereas timing channels involve using system resources to affect response time [37].[7]

Rather than forbid the existence of covert channels, the TCSEC specifies that systems should not contain covert channels of high bandwidth.[8] Low-bandwidth covert channels are allowed only because eliminating them is usually infeasible. And sometimes elimination is impossible: the proper function of some systems requires that some information be leaked. One example of such

[5]A *Trojan horse* [7] is a program that offers seemingly beneficial functionality, so that users will run the program—even if the program is given to them as a gift and they do not know its provenance or contents. But the program also contains malicious functionality of which users are unaware.

[6]Lampson also introduces "storage" and "legitimate" channels. The distinctions between these and covert channels—as Millen [89] observes—are somewhat elusive.

[7]Kemmerer [62] seems to be the source of TCSEC's categorization.

[8]The TCSEC defines "high" as 100 bits per second, the rate at which teletype terminals ran circa 1985. The "Light Pink Book" [91] offers a more nuanced analysis of what constitutes high bandwidth.

a system is a *password checker*, which allows or denies access to a system based on passwords supplied by users. By design, a password checker must release information about whether the passwords entered by users are correct.

Research into quantifying the bandwidth of covert channels began by employing *information theory*, the science of data transmission. Information theory could already quantify communication channel bandwidth, so its use with covert channels was natural. Denning's seminal work [35] in this area uses *entropy*, an information-theoretic metric for uncertainty, to calculate how much secret information can be leaked by a program. Millen [88] proposes *mutual information*, which is defined in terms of entropy, as a metric for information flow. These metrics make it possible to quantify information flow.

Much more history of computer security policies could be surveyed, but what we have covered suffices to put this dissertation in context. The beginning (a taxonomy of security policies) and the end (quantification of information flow) of our background are the places where this dissertation makes its contributions.

1.2 Contributions of this Dissertation

Quantification of security. Quantification of information flow is more difficult than at first it might seem. Consider a password checker PWC that sets an authentication flag a after checking a stored password p against a (guessed) password g supplied by the user.

$$PWC: \quad \textbf{if } p = g \textbf{ then } a := 1 \textbf{ else } a := 0$$

For simplicity, suppose that the password is either A, B, or C. Suppose also that the user is actually an attacker attempting to discover the password, and he believes the password is overwhelmingly likely to be A but has a minuscule and equally likely chance to be either B or C. (This need not be an arbitrary assumption on the attacker's part; perhaps the attacker was told by a usually reliable informant.) If the attacker experiments by executing PWC and guessing A, he expects to observe that a equals 1 upon termination. Such a confirmation of the attacker's belief would seem to convey some small amount of information. But suppose the informant was wrong: the real password is C. Then the attacker observes that a is equal to 0 and infers that A is not the password. Common sense dictates that his new belief is that B and C each have a 50% chance of being the password. The attacker's belief has greatly changed—he is surprised to discover the password is not A—so the outcome of this experiment conveys more information than the previous outcome. Thus, the information conveyed by executing PWC depends on what the attacker initially believed.

How much information flows from p to a in each of the above experiments? Answers to this question have traditionally been based on change in uncertainty, typically quantified by entropy or mutual information: information flow is quantified by the reduction in uncertainty about secret data [19, 24, 35, 49, 76, 82, 88]. Observe that, in the case where the password is C, the attacker initially is quite certain (though wrong) about the value of the password and after the experiment is rather uncertain about the value of the password; the change from "quite certain" to "rather uncertain" is an increase in uncertainty. So according to a metric based on reduction in uncertainty, no information flow occurred, which is anomalous and contradicts our intuition.

The problem with metrics based on uncertainty is twofold. First, they do not take *accuracy* into account. Accuracy and uncertainty are orthogonal properties of the attacker's belief—being certain does not make one correct—and as the password checking example illustrates, the amount of information flow depends on accuracy rather than on uncertainty. Second, uncertainty-based metrics are concerned with some unspecified agent's uncertainty rather than an attacker's. The unspecified agent is able to observe a probability distribution over secret input values but cannot observe the particular secret input used in the program execution. If the attacker were the unspecified agent, there would be no reason in general to assume that the probability distribution the attacker uses is correct. Because the attacker's probability distribution is therefore subjective, it must be treated as a belief. Beliefs are thus an essential—though until now uninvestigated—component of information flow.

Chapter 2 presents a new way to quantify information flow, based on these insights about beliefs and accuracy. We[9] give a formal model for *experiments*, which describe the interaction between attackers and systems by specifying how attackers update beliefs after observing system execution. This experiment model can be used with any mathematical representation of beliefs that supports three natural operations (product, update, and distance); as a concrete representation, we use probability distributions. Accordingly, we model systems as probabilistic imperative programs. We show that the result of belief update in the experiment model is equivalent to the attacker employing Bayesian inference, a standard technique in applied statistics for making inferences.

Our formula for calculating information flow is based on attacker beliefs before and after observing execution of a program. The formula is parameterized on the belief distance function; we make the formula concrete by instantiating it with *relative entropy*, which is an information-theoretic measure of the distance between two distributions. The resulting metric for the amount of *leakage* of se-

[9]Joint work with Andrew C. Myers and Fred B. Schneider.

cret information eliminates the anomaly described above, enabling quantification of information flow for individual executions of programs when attackers have subjective beliefs. We show that the metric correctly quantifies "information" as defined by information theory.[10] Moreover, we show that the metric generalizes previously defined uncertainty-based metrics.

Our metric also enables two kinds of analysis that were not previously possible. First, it is able to analyze *misinformation*, which is a negative information flow. We show that deterministic programs are incapable of producing misinformation. Second, our metric is able to analyze repeated interactions between an attacker and a system. This ability enables compositional reasoning about attacks—for example, about attackers who make a series of guesses in trying to determine a password.

We extend our experiment model to handle insiders, whose goal is to help the attacker learn secret information. Insiders are capable of influencing program execution, and we model them by introducing nondeterministic choice into programs. We show that if a program satisfies *observational determinism* [85, 102, 130], a noninterference policy for nondeterministic programs, then the quantity of information flow is always zero.

Previous work on quantitative information flow has considered only confidentiality, despite the fact that information theory itself is used to reason about integrity. Chapter 3 addresses this gap by applying the results of chapter 2 to integrity.[11] This application enables quantification of the amount of untrusted information with which an attacker can taint trusted information; we name this

[10]Information quantifies how surprising the occurrence of an event is. The *information* (or *self-information*) conveyed by an event is the negative logarithm of the probability of the event. An event that is certain (probability 1) thus conveys zero information, and as the probability decreases, the amount of information conveyed increases.

[11]Concurrent with the work described in this dissertation, Newsome et al. [94] also began to investigate quantitative information-flow integrity.

quantity *contamination*. Contamination is the information-flow dual of leakage, and it enjoys a similar interpretation based on information theory.

Moreover, our[12] investigation of information-flow integrity reveals another connection with information theory. Recall that information theory can be used to quantify the bandwidth, or channel capacity, of communication channels. We model such channels with programs that take trusted inputs from a *sender* and give trusted outputs to a *receiver*. The *transmission* of information to the receiver might be decreased because a program introduces random noise into its output that obscures the inputs, or because a program uses untrusted inputs (supplied by an attacker) in a way that obscures the trusted inputs. In either case, information is *suppressed*. We show how to quantify suppression; in expectation, this quantity is the same as the channel capacity. We analyze error-correcting codes [4] with suppression.

Simultaneously quantifying both confidentiality and integrity is also fruitful, because programs sometimes sacrifice integrity of information to improve confidentiality. For example, a statistical database that stores information about individuals might add randomly generated noise to a query response in an attempt to protect the privacy of those individuals. The addition of noise suppresses information yet reduces leakage, and our quantitative frameworks make this relationship precise: the amount of suppression plus the amount of leakage is a constant, for a given interaction between the database and a querier.

Formalization of security. The CIA taxonomy is an intuitive categorization of security requirements. Unfortunately, it is not supported by formal, mathematical theory: There is no formalization that simultaneously characterizes con-

[12]Joint work with Fred B. Schneider.

fidentiality, integrity, and availability.[13] Nor are confidentiality, integrity, and availability orthogonal—for example, the requirement that a principal be unable to read a value could be interpreted as confidentiality or unavailability of that value. And the CIA taxonomy provides little insight into how to enforce security requirements, because there is no verification methodology associated with any of the taxonomy's three categories.

This situation is similar to that of program verification circa the 1970s. Many specific properties of interest had been identified—for example, partial correctness, termination, and total correctness, mutual exclusion, deadlock freedom, starvation freedom, etc. But these properties were not all expressible in some unifying formalism, they are not orthogonal, and there was no verification methodology that was complete for all properties.

These problems were addressed by the development of the theory of trace properties. A *trace* is a sequence of execution states, and a *property* either holds or does not hold (i.e., is a Boolean function) of an object. Thus a *trace property* either holds or does not hold of an execution sequence. (The *extension* of a property is the set of objects for which the property holds. The extension of a property of individual traces—that is, a set of traces—sometimes is termed "property," too [5, 70]. But for clarity, "trace property" here denotes a set of traces.) Every trace property is the intersection of a safety property and a liveness property:

- A *safety property* is a trace property that proscribes "bad things" and can be proved using an invariance argument, and

[13]A formalism that comes close is that of Zheng and Myers [131], who define a particular noninterference policy for confidentiality, integrity, and availability.

- a *liveness property* is a trace property that prescribes "good things" and can be proved using a well-foundedness argument.[14]

This categorization forms an intuitively appealing and orthogonal basis from which all trace properties can be constructed. Moreover, safety and liveness properties are affiliated with specific, relatively complete verification methods. It is therefore natural to ask whether the theory of properties could be used to formalize security policies.

Unfortunately, important security policies cannot be expressed as properties of individual execution traces of a system [2, 44, 86, 103, 115, 117, 129]. For example, noninterference is not a property of individual traces, because whether a trace is allowed by the policy depends on whether another trace (obtained by deleting command executions by high users) is also allowed. For another example, stipulating a bound on mean response time over all executions is an availability policy that cannot be specified as a property of individual traces, because the acceptability of delays in a trace depends on the magnitude of delays in all other traces. However, both example policies are properties of systems, because a system (viewed as a whole, not as individual executions) either does or does not satisfy each policy.

The fact that security policies, like trace properties, proscribe and prescribe behaviors of systems suggested that a theory of security policies analogous to the theory of trace properties might exist. This dissertation develops that theory by formalizing security policies as properties of systems, or *system properties*. If systems are modeled as sets of execution traces, as with trace properties [70],

[14]Lamport [68] gave the first informal definitions of safety and liveness properties, appropriating the names from Petri net theory, and he also gave the first formal definition of safety [70]. Alpern and Schneider [5] gave the first formal definition of liveness and the proof that all trace properties are the intersection of safety and liveness properties; they later established the correspondence of safety to invariance and of liveness to well-foundedness [6].

then the extension of a system property is a set of sets of traces or, equivalently, a set of trace properties.[15] We[16] named this type of set a *hyperproperty* [29]. Every property of system behavior (for systems modeled as trace sets) can be specified as a hyperproperty, by definition. Thus, hyperproperties can describe trace properties and moreover can describe security policies, such as noninterference and mean response time, that trace properties cannot.

Chapter 4 shows that results similar to those from the theory of trace properties carry forward to hyperproperties:

- Every hyperproperty is the intersection of a safety hyperproperty and a liveness hyperproperty. (Henceforth, these terms are shortened to *hypersafety* and *hyperliveness*.) Hypersafety and hyperliveness thus form a basis from which all hyperproperties can be constructed.

- Hyperproperties from a class that we introduce, called *k-safety*, can be verified by using invariance arguments. Our verification methodology generalizes prior work on using invariance arguments to verify information-flow policies [12, 115].

However, we have not obtained complete verification methods for hypersafety or for hyperliveness.

The theory we develop also sheds light on the problematic status of refinement for security policies. Refinement never invalidates a trace property but can invalidate a hyperproperty: Consider a system π that nondeterministically chooses to output 0, 1, or the value of a secret bit h. System π satisfies the security policy "The possible output values are independent of the values of secrets." But one refinement of π is the system that always outputs h, and this

[15]McLean [86] gave the first formalization of security policies as properties of trace sets.
[16]Joint work with Fred B. Schneider.

system does not satisfy the security policy. We characterize the entire set of hyperproperties for which refinement is valid; this set includes the safety hyperproperties.

Safety and liveness not only form a basis for trace properties and hyperproperties, but they also have a surprisingly deep mathematical characterization in terms of topology. In the *Plotkin* topology on trace properties, safety and liveness are known to correspond to *closed* and *dense* sets, respectively [5]. We generalize this topological characterization to hyperproperties by showing that hypersafety and hyperliveness also correspond to closed and dense sets in a new topology, which turns out to be equivalent to the *lower Vietoris* construction applied to the Plotkin topology [109]. This correspondence could be used to bring results from topology to bear on hyperproperties.

Chapter 5 applies the theory of hyperproperties to models of system execution other than trace sets. We show that relational systems, labeled transition systems, state machines, and probabilistic systems all can be encoded as trace sets and handled using hyperproperties.

1.3 Dissertation Outline

Chapter 2 presents the new mathematical model and metric for quantitative information flow, as applied to confidentiality. Chapter 3 applies those ideas to integrity. Chapter 4 turns to the problem of a mathematical taxonomy of security policies and presents the results on hyperproperties. Chapter 5 extends those ideas to system models beyond trace sets. Related work is covered within each chapter. Chapter 6 concludes.

CHAPTER 2
QUANTIFICATION OF CONFIDENTIALITY*

Qualitative security properties, such as noninterference [46], typically either prohibit any flow of information from a high security level to a lower level, or they allow any information to flow provided it passes through some release mechanism. For a program whose correctness requires flow from high to low, the former policy is too restrictive and the latter can lead to unbounded leakage of information. Quantitative confidentiality policies, such as "at most k bits leak per execution of the program," allow information flows but at restricted rates. Such policies are useful when analyzing programs whose nature requires that some—but not too much—information be leaked, such as the password checker from chapter 1.

Recall that the amount of secret information a program leaks has traditionally been defined using change in uncertainty, but that definition leads to an anomaly when analyzing the password checker. We argued informally in chapter 1 that accuracy of beliefs provides a better explanation of the password checker. This chapter substantiates that argument with formal definitions and examples.

This chapter proceeds as follows. Basic representations for beliefs and programs are stated in §2.1. A model of the interaction between attackers and systems, describing how attackers update beliefs by observing execution of programs, is given in §2.2. A new quantitative flow metric, based on information theory, is defined in §2.3. The new metric characterizes the amount of information flow that results from change in the accuracy of an attacker's belief. The

*This chapter contains material from a previously published paper [28], which is © 2005 IEEE and reprinted with permission from *Proceedings of the 18th IEEE Computer Security Foundations Workshop*.

metric can also be instantiated to quantify change in uncertainty, and thus it generalizes previous information-flow metrics. The model and metric are formulated for use with any programming model that can be given a denotational semantics compatible with the representation of beliefs, as §2.4 illustrates with a particular programming language (**while**-programs plus probabilistic choice). The model is extended in §2.5 to programs in which nondeterministic choices are resolved by insiders, who are allowed to observe secret values. Related work is discussed in §2.6, and §2.7 concludes. Most proofs are delayed from the main body to appendix 2.A.

2.1 Incorporating Beliefs

A *belief* is a statement an agent makes about the state of the world, accompanied by some characterization of how certain the agent is about the truthfulness of the statement. Our agents will reason about probabilistic programs, so we begin by developing mathematical structures for representing programs and beliefs.

2.1.1 Distributions

A *frequency distribution* is a function δ that maps a program state to a *frequency*, which is a non-negative real number. A frequency distribution is essentially an unnormalized *probability distribution* over program states; it is easier to define a programming language semantics by using frequency distributions than by using probability distributions [101]. Henceforth, we write "distribution" to mean "frequency distribution."

The set of all program states is **State**, and the set of all distributions is **Dist**. The structure of **State** is mostly unimportant; it can be instantiated according to the needs of any particular language or system. For our examples, states map variables to values, where **Var** and **Val** are both countable sets:

$$\begin{aligned} v &\in \textbf{Var}, \\ \sigma \in \textbf{State} &\triangleq \textbf{Var} \to \textbf{Val}, \\ \delta \in \textbf{Dist} &\triangleq \textbf{State} \to \mathbb{R}^+. \end{aligned}$$

We write a state as a list of mappings—for example, $(g \mapsto A, a \mapsto 0)$ is a state in which variable g has value A and a has value 0.

The *mass* $\|\delta\|$ in a distribution δ is the sum of frequencies:[1]

$$\|\delta\| \quad \triangleq \quad (\textstyle\sum \sigma : \delta(\sigma)).$$

A probability distribution has mass 1, but a frequency distribution may have any non-negative mass. A *point mass* is a probability distribution that maps a single state to 1. It is denoted by placing a dot over that single state:

$$\dot{\sigma} \quad \triangleq \quad \lambda\sigma' \,.\, \text{if } \sigma' = \sigma \text{ then } 1 \text{ else } 0.$$

2.1.2 Programs

Execution of program S is described by a denotational semantics in which the meaning $[\![S]\!]$ of S is a function of type **State** $\to$ **Dist**. This semantics describes the frequency of termination in a given state: if $[\![S]\!]\sigma = \delta$, then the frequency that S terminates in σ' when begun in σ is $\delta(\sigma')$. This semantics can be lifted to a function of type **Dist** $\to$ **Dist** by the following definition:

$$[\![S]\!]\delta \quad \triangleq \quad (\textstyle\sum \sigma : \delta(\sigma) \cdot [\![S]\!]\sigma).$$

[1]Formula $(\star x \in D : R : P)$ is a quantification in which $\star$ is the quantifier (such as $\forall$ or Σ), x is the variable that is bound in R and P, D is the domain of x, R is the range, and P is the body. We omit D, R, and even x when they are clear from context; an omitted range means $R \equiv \mathit{true}$.

Thus, the meaning of S given a distribution on inputs is completely determined by the meaning of S given a state as input. By defining programs in terms of how they operate on distributions, we enable analysis of probabilistic programs.

Our examples use **while**-programs extended with a probabilistic choice construct. Let metavariables S, v, E, and B range over programs, variables, arithmetic expressions, and Boolean expressions, respectively. Evaluation of expressions is assumed side-effect free, but we do not otherwise prescribe their syntax or semantics. The syntax of the language is as follows:

$$\begin{array}{rcl} S & ::= & \textbf{skip} \mid v := E \mid S; S \mid \textbf{if } B \textbf{ then } S \textbf{ else } S \\ & & \mid \textbf{while } B \textbf{ do } S \mid S \ {}_{p}[\!] \ S \end{array}$$

The operational semantics for the deterministic subset of this language is standard. Probabilistic choice $S_1 \ {}_{p}[\!] \ S_2$ executes S_1 with probability p or S_2 with probability $1 - p$, where $0 \leq p \leq 1$. A denotational semantics for this language is given in §2.4.

2.1.3 Labels and Projections

We need a way to identify secret data; *confidentiality labels* serve this purpose. For simplicity, assume there are only two labels: a label L that indicates low-confidentiality (public) data, and a label H that indicates high-confidentiality (secret) data. Assume that **State** is a product of two domains $\textbf{State}_L$ and $\textbf{State}_H$, which contain the low- and high-labeled data, respectively. A *low state* is an element $\sigma_L \in \textbf{State}_L$; a *high state* is an element $\sigma_H \in \textbf{State}_H$. The projection of state $\sigma \in \textbf{State}$ onto $\textbf{State}_L$ is denoted $\sigma \upharpoonright L$; this is the part of σ visible to the attacker. Projection onto $\textbf{State}_H$, the part of σ not visible to the attacker, is denoted $\sigma \upharpoonright H$.

Each variable in a program is subscripted by a label to indicate the confidentiality of the information stored in that variable; for example, x_L is a variable that contains low information. For convenience, let variable l be labeled L and variable h be labeled H. $\mathbf{Var}_L$ is the set of variables in a program that are labeled L, so $\mathbf{State}_L = \mathbf{Var}_L \to \mathbf{Val}$. The *low projection* $\sigma \upharpoonright L$ of state σ is

$$\sigma \upharpoonright L \quad \triangleq \quad \lambda v \in \mathbf{Var}_L \,.\, \sigma(v).$$

States σ and σ' are *low-equivalent*, written $\sigma =_L \sigma'$, if they have the same low projection:

$$\sigma =_L \sigma' \quad \triangleq \quad (\sigma \upharpoonright L) \;=\; (\sigma' \upharpoonright L).$$

Distributions also have projections. Let δ be a distribution and σ_L a low state. Then $(\delta \upharpoonright L)(\sigma_L)$ is the combined frequency of those states whose low projection is σ_L:

$$\delta \upharpoonright L \quad \triangleq \quad \lambda \sigma_L \in \mathbf{State}_L \,.\, (\textstyle\sum \sigma' \;:\; (\sigma' \upharpoonright L) \;=\; \sigma_L \;:\; \delta(\sigma')).$$

High projection and high equivalence are defined by replacing occurrences of L with H in the definitions above.

2.1.4 Belief Representation

To be usable in our framework, a belief representation must support certain natural operations. Let b and b' be beliefs ranging over sets of possible worlds W and W', respectively, where a *possible world* is some elementary outcome about which beliefs can be held [52].

1. *Belief product* $\otimes$ combines b and b' into a new belief $b \otimes b'$ about possible worlds $W \times W'$, where W and W' are disjoint.

2. *Belief update* $b|U$ is the belief that results when b is updated to include new information that the actual world is in a set $U \subseteq W$ of possible worlds.
3. *Belief distance* $D(b \rightarrow b')$ is a real number $r \geq 0$ that quantifies differences between b and b'.

Although the results in this chapter are, for the most part, independent of any particular representation, the rest of this chapter uses distributions to represent beliefs. High states are the possible worlds for beliefs, and a belief is a probability distribution over high states:

$$b \in \textbf{Belief} \quad \triangleq \quad \textbf{State}_H \rightarrow \mathbb{R}^+, \quad \text{s.t. } \|b\| = 1.$$

Thus, beliefs correspond to probability measures. Probability measures are well-studied as a belief representation [52], and they have several advantages here: they are familiar, quantitative, support the operations required above, and admit a programming language semantics (as shown in §2.4). There is also a nice justification for the numbers they produce: roughly, $b(\sigma)$ characterizes the amount of money an attacker should be willing to bet that σ is the actual state of the system [52]. Other choices of belief representation could include belief functions or sets of probability measures [52]. Although these alternatives are more expressive than probability measures, it is more complicated to define the required operations for them.

For belief product $\otimes$, we employ a distribution product $\otimes$ of two distributions $\delta_1 : A \rightarrow \mathbb{R}^+$ and $\delta_2 : B \rightarrow \mathbb{R}^+$, with A and B disjoint:

$$\delta_1 \otimes \delta_2 \quad \triangleq \quad \lambda(\sigma_1, \sigma_2) \in A \times B \,.\, \delta_1(\sigma_1) \cdot \delta_2(\sigma_2).$$

It is easy to check that if b and b' are beliefs, $b \otimes b'$ is too.

For belief update $|$, we use *distribution conditioning*:

$$\delta|U \quad \triangleq \quad \lambda\sigma \,.\, \text{if } \sigma \in U \text{ then } \frac{\delta(\sigma)}{(\sum \sigma' \in U \,:\, \delta(\sigma'))} \text{ else } 0.$$

For belief distance D we use *relative entropy*, an information-theoretic metric [59] for the distance between distributions:

$$D(b \rightarrow b') \quad \triangleq \quad (\sum \sigma \,:\, b'(\sigma) \cdot \log \tfrac{b'(\sigma)}{b(\sigma)}).$$

The base of the logarithm in D can be chosen arbitrarily; we use base 2 and write $\lg$ to indicate $\log_2$, making bits the unit of measurement for distance. The relative entropy of b to b' is the expected inefficiency (that is, the number of additional bits that must be sent) of an optimal code that is constructed by assuming an inaccurate distribution over symbols b when the real distribution is b' [32]. Like an analytic metric, $D(b \rightarrow b')$ is always at least zero and $D(b \rightarrow b')$ equals zero only when $b = b'$.[2]

Relative entropy has the property that if $b'(\sigma) > 0$ and $b(\sigma) = 0$, then $D(b \rightarrow b') = \infty$. Intuitively, b' is "infinitely surprising" because it regards σ as possible whereas b regards σ as impossible. To avoid this anomaly, beliefs may be required to satisfy an *admissibility restriction*, which ensures that attackers do not initially believe that certain states are impossible. For example, a belief might be restricted such that it never differs by more than a factor of ϵ from a uniform distribution. This restriction could be useful with the password checker (c.f. §1.2) if it is reasonable to assume that attackers believe that all passwords are nearly equally likely. Or, the attacker's belief may be required to be a maximum entropy distribution [32] with respect to attacker-specified constraints. This restriction could be useful with the password checker if attackers believe that passwords are English words (which is a kind of constraint). Other admissibility restrictions can be substituted for these when stronger assumptions can be made about attacker beliefs.

[2]Unlike an analytic metric, D does not satisfy symmetry or the triangle inequality. However, it seems unreasonable to assume that either of these properties holds for beliefs, since it can be easier to rule out a possibility from a belief than to add a new possibility, or vice-versa.

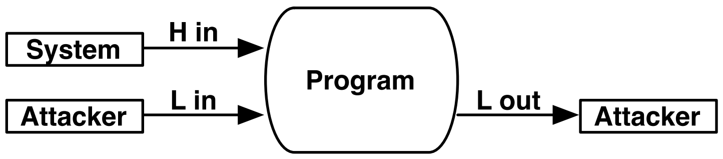

Figure 2.1: Channels in confidentiality experiment

2.2 Confidentiality Experiments

We formalize as a *confidentiality experiment* (or simply an *experiment*) how an *attacker*, an agent that reasons about secret data, revises his beliefs from interaction with program that is executed by a *system*. The attacker should not learn about the high input to the program but is allowed to observe and influence low inputs and outputs. Other agents (a system operator, other users of the system with their own high data, an informant upon which the attacker relies, etc.) might be involved when an attacker interacts with a system; however, it suffices to condense all of these to just the attacker and the system. The channels between agents and the program are depicted in figure 2.1 and are described in detail below.

We conservatively assume that the attacker knows the code of the program with which he interacts. For simplicity, we assume that the program always terminates and that it never modifies the high state. Both restrictions can be lifted without significant changes, as shown in §2.2.4.

2.2.1 Confidentiality Experiment Protocol

Formally, an experiment $\mathcal{E}$ is described by a tuple,

$$\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle,$$

An experiment $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle$ is conducted as follows.

1. The attacker chooses a prebelief b_H about the high state.
2. (a) The system picks a high state σ_H.

   (b) The attacker picks a low state σ_L.
3. The attacker predicts the output distribution: $\delta'_A = [\![S]\!](\dot{\sigma}_L \otimes b_H)$.
4. The system executes program S, which produces a state $\sigma' \in \delta'$ as output, where $\delta' = [\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H)$. The attacker observes the low projection of the output state: $o = \sigma' \upharpoonright L$.
5. The attacker infers a postbelief: $b'_H = (\delta'_A | o) \upharpoonright H$.

Figure 2.2: Experiment protocol

where S is the program, b_H is the attacker's belief at the beginning of the experiment, σ_H is the high projection of the initial state, and σ_L is the low projection of the initial state. The protocol for experiments, which uses some notation defined below, is summarized in figure 2.2. Here is a justification for the protocol.

An attacker's *prebelief* b_H, describing his belief at the beginning of the experiment (step 1), may be chosen arbitrarily (subject to an admissibility restriction as in §2.1.4) or may be informed by previous experiments. In a series of experiments, the *postbelief* from one experiment typically becomes the prebelief to the next. The attacker might even choose a prebelief b_H that contradicts his true subjective probability distribution for the state, and this gives our analysis additional power by allowing the attacker to conduct experiments to answer questions such as "What would happen if I were to believe b_H?"

The system chooses σ_H (step 2a), the high projection of the initial state, and this part of the state might remain constant from one experiment to the next or might vary. For example, Unix passwords do not usually change frequently, but the output displayed on an RSA SecurID token changes each minute. We conservatively assume that the attacker chooses all of σ_L (step 2b), the low projection of the initial state. This gives the attacker additional power in controlling execution of the program, which he can use to attempt to maximize the amount of information flow. The attacker's choice of σ_L is thus likely to be influenced by b_H, but for generality, we do not require there be such a strategy.

Using the semantics of S along with prebelief b_H as a distribution on high input, the attacker conducts a "thought experiment" to generate a *prediction* of the output distribution (step 3). We define prediction δ'_A to correlate the output state with the high input state:

$$\delta'_A = [\![S]\!](\dot{\sigma}_L \otimes b_H).$$

Program S is executed (step 4) only once in each experiment; multiple executions are modeled by multiple experiments. The meaning of S given inputs σ_L and σ_H is an output distribution δ':

$$\delta' = [\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H).$$

From δ' the attacker makes an *observation*, which is a low projection of an output state. Probabilistic programs may yield many possible output states, but in a single execution of the program, only one output state is actually produced. This output state σ' is produced with frequency $\delta'(\sigma')$. We write $\sigma' \in \delta'$ to denote that σ' is in the support of (i.e., has positive frequency according to) δ'. In a single experiment, the attacker is allowed only a single observation. The observation o resulting from σ' is $\sigma' \upharpoonright L$.

Finally, the attacker incorporates any new inferences that can be made from observation o by conditioning prediction δ'_A. The result is projected to H to produce the attacker's postbelief b'_H (step 5):

$$b'_H = (\delta'_A | o) \upharpoonright H.$$

Here, conditioning operator $\delta|o$ is defined in terms of conditioning operator $\delta|U$. The new operator removes all mass in distribution δ that is inconsistent with observation o, then normalizes the result:

$$\begin{aligned} \delta|o &\triangleq \delta|\{\sigma' \mid \sigma' \upharpoonright L = o\} \\ &= \lambda\sigma \,.\, \text{if } (\sigma \upharpoonright L) \;=\; o \text{ then } \tfrac{\delta(\sigma)}{(\delta|L)(o)} \text{ else } 0. \end{aligned}$$

2.2.2 Password Checking as an Experiment

Our experiment model allows the informal reasoning in §1.2 to be made precise. For example, consider the password checker; adding confidentiality labels yields:

$$PWC: \quad \textbf{if } p_H = g_L \textbf{ then } a_L := 1 \textbf{ else } a_L := 0$$

The attacker begins an experiment by choosing prebelief b_H, perhaps as specified in the column labeled b_H of table 2.1. Next, the system chooses initial high projection σ_H, and the attacker chooses initial low projection σ_L. In the first experiment in §1.2, the password was A, so the system chooses $\sigma_H = (p \mapsto A)$. Similarly, the attacker chooses $\sigma_L = (g \mapsto A, a \mapsto 0)$. (The initial value of a is actually irrelevant, since it is never used by the program and a is set along all control paths.) Next, the system executes PWC. Output distribution δ' should be the point mass at state $\sigma' = (p \mapsto A, g \mapsto A, a \mapsto 1)$; the semantics in §2.4 will validate this intuition. Since σ' is the only state that can be sampled from δ', the attacker's observation o_1 is $\sigma' \upharpoonright L = (g \mapsto A, a \mapsto 1)$.

Finally, the attacker infers a postbelief. He conducts a thought experiment, predicting an output distribution $\delta'_A = [\![PWC]\!](\dot{\sigma}_L \otimes b_H)$, given in table 2.2. The ellipsis in the final row of the table indicates that all states not shown have frequency 0. This distribution is intuitively correct: the attacker believes that he has a 98% chance of being authenticated, whereas 1% of the time he will fail to

Table 2.1: Beliefs about p_H

p_H	b_H	b'_{H1}	b'_{H2}
A	0.98	1	0
B	0.01	0	0.5
C	0.01	0	0.5

Table 2.2: Distributions on PWC output

p	g	a	δ'_A	$\delta'_A \vert o_1$	$\delta'_A \vert o_2$
A	A	0	0	0	0
A	A	1	0.98	1	0
B	A	0	0.01	0	0.5
B	A	1	0	0	0
C	A	0	0.01	0	0.5
C	A	1	0	0	0
	...		0	0	0

be authenticated because the password is B, and another 1% because it is C. The attacker conditions prediction δ'_A on observation o_1, obtaining $\delta'_A|o_1$, also shown in table 2.2. Projecting to high yields the attacker's postbelief, b'_{H1}, shown in table 2.1. This postbelief is what the informal reasoning in §1.2 suggested: the attacker is certain that the password is A.

The second experiment in §1.2 can also be formalized. In it, b_H and σ_L remain the same as before, but σ_H becomes $(p \mapsto C)$. Observation o_2 is therefore the point mass at $(g \mapsto A, a \mapsto 0)$. Prediction δ'_A remains unchanged, and conditioned on o_2 it becomes $\delta'_A|o_2$, shown in table 2.2. Projecting to high yields postbelief b'_{H2} from table 2.1. This postbelief again agrees with the informal reasoning: the attacker believes that there is a 50% chance each for the password to be B or C.

2.2.3 Bayesian Belief Revision

The formula the attacker uses to infer a postbelief is an application of Bayesian inference, which is a standard technique used in applied statistics for making inferences when uncertainty is made explicit through probability models [45]. The attacker therefore reasons rationally, according to Halpern's rationality axioms [52], though the literature on human behavior shows that this is not always the same as human reasoning [60, 64].

Let belief revision operator $\mathcal{B}$ yield the postbelief from an experiment $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle$, given observation o:

$$\mathcal{B}(\mathcal{E}, o) \quad \triangleq \quad (\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H)|o) \upharpoonright H.$$

We write $b'_H \in \mathcal{B}(\mathcal{E})$ to denote that there exists some o for which $b'_H = \mathcal{B}(\mathcal{E}, o)$.

Recall Bayes' rule for updating a hypothesis Hyp with an observation obs:

$$\Pr(Hyp|obs) \quad = \quad \frac{\Pr(Hyp)\Pr(obs|Hyp)}{(\sum Hyp' : \Pr(Hyp')\Pr(obs|Hyp'))}.$$

In our model, the attacker's hypothesis is about the values of high states, so the domain of hypotheses is **State** $\upharpoonright H$. Therefore $\Pr(Hyp)$, the probability the attacker ascribes to a particular hypothesis, is modeled by $b_H(\sigma_H)$. The probability $\Pr(obs|Hyp)$ the attacker ascribes to an observation given the assumed truth of a hypothesis is modeled by the program semantics: the probability of observation o given an assumed high input σ_H is $(\llbracket S \rrbracket(\dot{\sigma}_L \otimes \dot{\sigma}_H) \upharpoonright L)(o)$.

Given experiment $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle$, instantiating Bayes' rule on these probabilities yields Bayesian inference $BI(\mathcal{E}, o)$, which is $\Pr(\sigma_H|o)$:

$$BI(\mathcal{E}, o) = \frac{b_H(\sigma_H) \cdot (\llbracket S \rrbracket(\dot{\sigma}_L \otimes \dot{\sigma}_H) \upharpoonright L)(o)}{(\sum \sigma'_H : b_H(\sigma'_H) \cdot (\llbracket S \rrbracket(\dot{\sigma}_L \otimes \dot{\sigma}'_H) \upharpoonright L)(o))}.$$

With this instantiation, we can show that the experiment protocol leads an attacker to update his belief according to Bayesian inference:

Theorem 2.1. $\mathcal{B}(\mathcal{E}, o)(\sigma_H) = BI(\mathcal{E}, o)$.

Proof. In appendix 2.A. $\square$

2.2.4 Mutable High Inputs and Nontermination

Two simplifying assumptions about programs were invoked by §2.2.1: programs never modify high input, and they always terminate. We now dispense with these technical issues.

Mutable high inputs. If program S were to modify the high state, the attacker's prediction δ'_A would correlate high outputs with low outputs. However, to calculate a postbelief (in step 5), δ'_A must correlate high *inputs* with low outputs. So our experiment protocol requires the high input state be preserved in δ'_A.

Informally, we can do this by keeping a copy of the initial high inputs in the program state. This copy is never modified by the program. Thus, the copy is preserved in the final output state, and the attacker can again establish a correlation between high inputs and low outputs.

Formally, let the notation b^0_H mean the same distribution as b_H, except that each state of its domain has a 0 as a superscript. So, if b_H ascribes probability p to state σ, then b^0_H ascribes probability p to the state σ^0. We assume that S cannot modify states with a superscript 0. In the case that states map variables to values, this could be achieved by defining σ^0 to be the same state as σ, but with the superscript 0 attached to variables; for example, if $\sigma(v) = 1$ then $\sigma^0(v^0) = 1$. Note that S cannot modify σ^0 if did not originally contain any variables with superscripts.

Using this notation, the belief revision operator is extended to $\mathcal{B}^!$, which allows S to modify the high state in experiment $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle$:

$$\mathcal{B}^!(\mathcal{E}, o) \;\triangleq\; ((\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H \otimes b_H^0)|o)) \restriction H^0.$$

In this definition, the high input state is preserved by introducing the product with b_H^0, and the attacker's postbelief about the input is recovered by restricting to H^0, the high input state with the superscript 0.

Nontermination. To eliminate the second assumption, note that program S must terminate for an attacker to obtain a low state as an observation when executing S. If the attacker has an oracle that decides nontermination,[3] then nontermination can be modeled in the standard denotational style with a state $\perp$ representing divergence, as follows.

Let $\textbf{State}_\perp \triangleq \textbf{State} \cup \{\perp\}$, and $\perp \restriction L \triangleq \perp$. Nontermination is now allowed as an observation, leading to an extended belief revision operator $\mathcal{B}^{!\perp}$:

$$\mathcal{B}^{!\perp}(\mathcal{E}, o) \;\triangleq\; (out_\perp(S, \dot{\sigma}_L \otimes b_H \otimes b_H^0)|o) \restriction H^0.$$

[3]An attacker that cannot detect nontermination is more difficult to model. At some point during the execution of the program, he can stop waiting for the program to terminate and declare that he has observed nontermination. However, he might be incorrect in doing so—leading to beliefs about nontermination and instruction timings. The interaction of these beliefs with beliefs about high inputs would be complex; we do not address it here.

Observation o is now produced from output distribution $\delta' = out_\perp(S, \dot{\sigma}_L \otimes \dot{\sigma}_H)$. Function $out_\perp(S, \delta)$ produces a distribution which yields the frequency that S terminates, or fails to terminate, on input distribution δ:

$$\begin{aligned} out_\perp(S,\delta) \;&\triangleq\; \lambda\sigma : \textbf{State}_\perp \,.\, \text{if } \sigma = \perp \\ &\qquad\quad \text{then } \|\delta\| - \|[\![S]\!]\delta\| \\ &\qquad\quad \text{else } ([\![S]\!]\delta)(\sigma). \end{aligned}$$

If S does not terminate on some input states in δ, output distribution $[\![S]\!]\delta$ will contain less mass than δ; otherwise, $\|\delta\| = \|[\![S]\!]\delta\|$. Missing mass corresponds to nontermination [83, 101], so $out_\perp$ maps the missing mass to $\perp$.

2.3 Quantification of Information Flow

The informal analysis of PWC in §1.2 suggests that information flow corresponds to an improvement in the accuracy of an attacker's belief. We now formalize that analysis by using change in accuracy, as measured by belief distance D, to quantify information flow.

2.3.1 Information Flow from an Outcome

Given an experiment $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle$, an *outcome* is a postbelief b'_H such that $b'_H \in \mathcal{B}(\mathcal{E})$, where $\mathcal{B}$ is the belief revision operator from §2.2.3. Recall from §2.1.4 that $D(b \rightarrow b')$ is the distance from belief b to belief b'. The accuracy of the attacker's prebelief b_H in experiment $\mathcal{E}$ is $D(b_H \rightarrow \dot{\sigma}_H)$; the accuracy of outcome b'_H, the attacker's postbelief, is $D(b'_H \rightarrow \dot{\sigma}_H)$.

We define the amount of information flow $\mathcal{Q}$ caused by outcome b'_H of experiment $\mathcal{E}$ as the difference of those two quantities:

$$\mathcal{Q}(\mathcal{E}, b'_H) \quad \triangleq \quad D(b_H \rightarrow \dot{\sigma}_H) - D(b'_H \rightarrow \dot{\sigma}_H).$$

Thus quantity of flow $\mathcal{Q}$ is the improvement in the accuracy of the attacker's belief. This amount can positive or negative; we defer discussion of negative flow to §2.3.3. Since D is instantiated with relative entropy, the unit of measurement for $\mathcal{Q}$ is (information-theoretic) bits.

With an additional definition from information theory, a more consequential characterization of $\mathcal{Q}$ is possible. Let $\mathcal{I}_\delta(F)$ denote the *information* contained in event F drawn from probability distribution δ:

$$\mathcal{I}_\delta(F) \quad \triangleq \quad -\lg \Pr_\delta(F).$$

Information is sometimes called "surprise" because $\mathcal{I}$ quantifies how surprising an event is; for example, when an event that has probability 1 occurs, no information (0 bits) is conveyed because the occurrence is completely unsurprising.

For an attacker, the outcome of an experiment involves two unknowns: the initial high state σ_H and the probabilistic choices made by the program. Let $\delta_S = [\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H) \upharpoonright L$ be the system's distribution on low outputs, and $\delta_A = [\![S]\!](\dot{\sigma}_L \otimes b_H) \upharpoonright L$ be the attacker's distribution on low outputs. $\mathcal{I}_{\delta_A}(o)$ quantifies the information contained in o about both unknowns, but $\mathcal{I}_{\delta_S}(o)$ quantifies only the probabilistic choices made by the program.[4] For programs that make no probabilistic choices, δ_A contains information about only the initial high state, and δ_S is a point mass at some state σ such that $\sigma \upharpoonright L = o$. So amount of information $\mathcal{I}_{\delta_S}(o)$ is 0. For probabilistic programs, $\mathcal{I}_{\delta_S}(o)$ is generally not

[4]The technique used in §2.2.4 for modeling nontermination ensures that δ_A and δ_S are probability distributions. Thus, $\mathcal{I}_{\delta_A}$ and $\mathcal{I}_{\delta_S}$ are well-defined.

equal to 0; subtracting it removes all the information contained in $\mathcal{I}_{\delta_A}(o)$ that is solely about the results of probabilistic choices, leaving information only about high inputs.

The following theorem states that $\mathcal{Q}$ quantifies the information about high input σ_H contained in observation o:

Theorem 2.2. $\mathcal{Q}(\mathcal{E}, b'_H) = \mathcal{I}_{\delta_A}(o) - \mathcal{I}_{\delta_S}(o)$.

Proof. In appendix 2.A. $\square$

As an example, consider the experiments involving PWC in §2.2.2. The first experiment $\mathcal{E}_1$ has the attacker correctly guess the password A, so

$$\mathcal{E}_1 = \langle PWC, b_H, (p \mapsto A), (g \mapsto A, a \mapsto 0)\rangle,$$

where table 2.1 defines b_H (and the other beliefs used below). Only one outcome, b'_{H1}, is possible from this experiment. We calculate the amount of flow from this outcome, letting $\sigma_H = (p \mapsto A)$:

$$\begin{aligned}
\mathcal{Q}(\mathcal{E}_1, b'_{H1}) &= D(b_H \rightarrow \dot{\sigma}_H) - D(b'_{H1} \rightarrow \dot{\sigma}_H) \\
&= (\sum \sigma'_H \;:\; \dot{\sigma}_H(\sigma'_H) \cdot \lg \tfrac{\dot{\sigma}_H(\sigma'_H)}{b_H(\sigma'_H)}) \;-\; (\sum \sigma'_H \;:\; \dot{\sigma}_H(\sigma'_H) \cdot \lg \tfrac{\dot{\sigma}_H(\sigma'_H)}{b'_{H1}(\sigma'_H)}) \\
&= -\lg b_H(\sigma_H) + \lg b'_{H1}(\sigma_H) \\
&= 0.0291
\end{aligned}$$

This small flow makes sense because the outcome has only confirmed something the attacker already believed to be almost certainly true. In experiment $\mathcal{E}_2$ the attacker guesses incorrectly:

$$\mathcal{E}_2 = \langle PWC, b_H, (p \mapsto C), (g \mapsto A, a \mapsto 0)\rangle.$$

Again, only one outcome is possible from this experiment, and calculating $\mathcal{Q}(\mathcal{E}_2, b'_{H2})$ yields an information flow of 5.6439 bits. This higher information flow makes sense, because the attacker's postbelief is much closer to correctly identifying the high state. The attacker's prebelief b_H ascribed a 0.02 probability to the event $p \neq A$, and the information conveyed by an event with probability 0.02 is 5.6439. This suggests that $\mathcal{Q}$ is the right metric for the information about high input contained in the observation.

The information flow of 5.6439 bits in experiment $\mathcal{E}_2$ might seem surprisingly high. At most two bits are required to store password p in memory, so why does the program leak more than five bits? Here, the greater leakage occurs because the attacker's belief is not uniform. A uniform prebelief (ascribing $1/3$ probability to each password A, B, and C) would, in a series of experiments, cause the attacker to learn a total of $\lg 3 \approx 1.6$ bits. However, belief b_H is more erroneous than the uniform belief, so a larger amount of information is required to correct it.

An uncertainty-based definition for information flow does not produce a reasonable leakage for this experiment. The attacker's initial uncertainty about p is $\mathcal{H}(b_H) = 0.1614$ bits, where $\mathcal{H}$ is the information-theoretic metric of *entropy*, or uncertainty, in a probability distribution δ:

$$\mathcal{H}(\delta) \quad \triangleq \quad -(\textstyle\sum \sigma \, : \, \delta(\sigma) \cdot \lg \delta(\sigma)).$$

In the second experiment, the attacker's final uncertainty about p is $\mathcal{H}(b_{H2}) = 1$. The reduction in uncertainty is $0.1614 - 1 = -0.8386$, hence there is actually an increase in uncertainty. So the uncertainty-based analysis that we have performed is forced to conclude that information did not flow to the attacker. But this is clearly not the case—the attacker's belief has been guided closer to reality by the experiment. The uncertainty-based analysis ignores reality by comparing b_H and b_{H2} against themselves, instead of against the high state σ_H.

2.3.2 Interpreting Metric $\mathcal{Q}$

According to theorem 2.2, metric $\mathcal{Q}$ correctly quantifies the amount of information flow, in bits. But what does it mean to leak one bit of information? The next theorem states that k bits of leakage correspond to a k-fold doubling of the probability that the attacker ascribes to reality.

Theorem 2.3. *Let* $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle$*. Then:*

$$\mathcal{Q}(\mathcal{E}, b'_H) = k \;\equiv\; b'_H(\sigma_H) = 2^k \cdot b_H(\sigma_H).$$

Proof. In appendix 2.A. □

Suppose an attacker were to guess what reality is by sampling from his belief b_H; the probability he guesses correctly is $b_H(\sigma_H)$. Thus, by theorem 2.3, one bit of leakage makes the attacker twice as likely to guess correctly. This reveals an interesting analogy with the uncertainty-based definition. In it, one bit of leakage corresponds to the attacker becoming twice as certain about the high state, though he may, as the example in §2.3.1 shows, become certain about the wrong high state. However, one bit of leakage in our accuracy-based definition corresponds to the attacker becoming twice as certain about the *correct* high state.

2.3.3 Accuracy, Uncertainty, and Misinformation

Accuracy and uncertainty are orthogonal properties of beliefs, as depicted in figure 2.3. The figure shows the change in an attacker's accuracy and uncertainty when the program

$$\mathit{FLIP}: \quad l := h \;{}_{0.99}\!\left[\right]\; l := \neg h$$

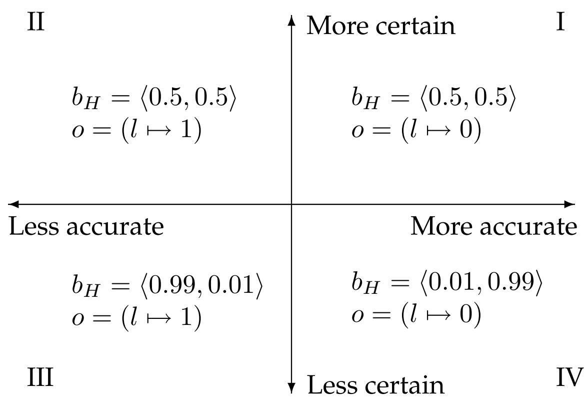

Figure 2.3: Effect of *FLIP* on postbelief

Table 2.3: Analysis of *FLIP*

	Quadrant			
	I	II	III	IV
$b_H(h \mapsto 0)$	0.5	0.5	0.99	0.01
$b_H(h \mapsto 1)$	0.5	0.5	0.01	0.99
o	$(l \mapsto 0)$	$(l \mapsto 1)$	$(l \mapsto 1)$	$(l \mapsto 0)$
$b'_H(h \mapsto 0)$	0.99	0.01	0.5	0.5
$b'_H(h \mapsto 1)$	0.01	0.99	0.5	0.5
Increase in accuracy	$+0.9855$	-5.6439	-0.9855	$+5.6439$
Reduction in uncertainty	$+0.9192$	$+0.9192$	-0.9192	-0.9192

is analyzed with experiment $\mathcal{E} = \langle \mathit{FLIP}, b_H, (h \mapsto 0), (l \mapsto 0) \rangle$ and observation o is generated by the experiment. The notation $b_H = \langle x, y \rangle$ in figure 2.3 means that $b_H(h \mapsto 0) = x$ and $b_H(h \mapsto 1) = y$.

Usually, *FLIP* sets l to be h, so the attacker will expect this to be the case. Executions in which this occurs will cause his postbelief to be more accurate, but may cause his uncertainty to either increase or decrease, depending on his prebelief; when uncertainty increases, an uncertainty metric would mistakenly say that no flow has occurred.

With probability 0.01, *FLIP* produces an execution that fools the attacker and sets l to be $\neg h$, causing his belief to become less accurate. The decrease in accuracy results in *misinformation*, which is a negative information flow. When the attacker's prebelief is almost completely accurate, such executions will make him more uncertain. But when the attacker's prebelief is uniform, executions that result in misinformation will make him less uncertain; when uncertainty decreases, an uncertainty metric would mistakenly say that flow has occurred.

Table 2.3 concretely demonstrates the orthogonality of accuracy and uncertainty. The quadrant labels refer to figure 2.3. The attacker's prebelief b_H, observation o, and resulting postbelief b'_H are given in the top half of the table. In the bottom half of the table, increase in accuracy is calculated using information flow metric $\mathcal{Q}$, and reduction in uncertainty is calculated using the difference in entropy $\mathcal{H}(b_H) - \mathcal{H}(b'_H)$. The symmetries in the bottom half of the table are a result of the symmetries between prebeliefs and postbeliefs. Quadrants II and IV, for example, have exchanged these beliefs, which for both metrics has the effect of negating the amount of information flow.

The probabilistic choice in $FLIP$ is essential for producing misinformation, as shown by the following theorem. Let **Det** be the set of syntactically deterministic programs, i.e., programs that do not contain any probabilistic choice. Because they lack a source of randomness, these programs cannot decrease the accuracy of an attacker's belief:

Theorem 2.4. $S \in \textbf{Det} \implies \forall \mathcal{E}, b'_H \in \mathcal{B}(\mathcal{E}) \,.\, \mathcal{Q}(\mathcal{E}, b'_H) \geq 0.$

Proof. In appendix 2.A. □

2.3.4 Emulating Uncertainty

The accuracy metric of §2.3.1 generalizes uncertainty metrics. Informally, this is because uncertainty metrics recognize only two distributions (belief before and after execution), whereas our framework recognizes these plus one additional distribution (reality). By ignoring reality, our framework can produce the same results as many uncertainty metrics. Here we show how to emulate the metric of Clark et al. [25].

Let A, B, and C be random variables. The *conditional mutual information* $\mathcal{I}(A, B|C)$ is the amount of uncertainty about the value of A that is resolved by learning the value of B, given prior knowledge of the value of C [32]. Conditional mutual information is defined using a generalization of the entropy function from §2.3.1 to conditional entropy [32]:

$$\begin{aligned}\mathcal{I}(A, B|C) &\triangleq \mathcal{H}(A|C) - \mathcal{H}(A|B, C)\\ &= \sum_a \sum_b \sum_c \Pr(a, b, c) \lg \frac{\Pr(a, b|c)}{\Pr(a|c) \cdot \Pr(b|c)}\,.\end{aligned}$$

In this definition, a abbreviates $A = a$, etc. The probability is taken with respect to the joint distribution on A, B, and C.

The metric of Clark et al. states that the amount of information flow $\mathcal{L}$ from high input H_{in} into low output L_{out}, given low input L_{in},[5] is the mutual information between H_{in} and L_{out}, given L_{in}:

$$\mathcal{L}(H_{in}, L_{in}, L_{out}) \quad\triangleq\quad \mathcal{I}(H_{in}, L_{out}|L_{in}).$$

First, to instantiate our framework to that of Clark et al., we force our framework to ignore reality by introducing an admissibility restriction (c.f. §2.1.4): prebeliefs must be identical to the system's chosen high input distribution. This means that prebeliefs must be correct; there can be no error in the attacker's estimate of the probability distribution on high inputs.

Second, we adjust the definition of belief. The uncertainty model of Clark et al. calculates information flow as an expectation over a probability distribution

[5]Their metric more generally allows the quantification of information flow into any subset of the output variables. The approach we give here can similarly be generalized.

on both low and high inputs. We could model this using the techniques about to be introduced in §2.3.5 and §2.3.6, but because of the admissibility restriction just made, it is equivalent and simpler to allow beliefs to range over low state as well as high state. As before, we assume that high state remains constant using the copying technique of §2.2.4. Since beliefs now include low state, we must also apply this technique to assure that the initial values of low variables are preserved in the state. Let the low input component of the state be denoted L^0. Assume that the attacker's prebelief b ranges over $L^0 \cup H^0$, whereas his postbelief b' ranges over $L^0 \cup H^0 \cup L \cup H$.

We want to establish that accuracy metric $\mathcal{Q}$ yields the same result as uncertainty metric $\mathcal{L}$ for any outcome. Recall that $\mathcal{Q}$ is defined in terms of distance function D. Our previous instantiation of D as relative entropy yielded an accuracy metric. Now we reinstantiate D using (non-relative) entropy:

$$D(b \twoheadrightarrow b') \quad = \quad \mathcal{H}(b \restriction (L \cup L^0 \cup H^0)) - \mathcal{H}(b \restriction (L \cup L^0)).$$

Observe that this instantiation ignores argument b', the belief representing reality. Let $H_{in} = \dot{\sigma}_H$, $L_{in} = \dot{\sigma}_L$, and $L_{out} = \delta' \restriction L$, where δ' is the output distribution from the experiment protocol. This yields that amount of information flow $\mathcal{Q}$ is the same as uncertainty metric $\mathcal{L}$:

Theorem 2.5. $\mathcal{Q}(\mathcal{E}, b') = \mathcal{L}(H_{in}, L_{in}, L_{out})$.

Proof. In appendix 2.A. □

We discuss another relationship between accuracy and uncertainty in §2.3.6.

2.3.5 Expected Flow for an Experiment

Since an experiment on a probabilistic program can produce many observations, and therefore many outcomes, it is desirable to characterize expected flow over those outcomes. So we define expected flow $\mathcal{Q}_{\mathsf{E}}$ over all observations from experiment $\mathcal{E}$:

$$\begin{array}{rcl} \mathcal{Q}_{\mathsf{E}}(\mathcal{E}) & \triangleq & \mathsf{E}_{o \in \delta' \restriction L}[\mathcal{Q}(\mathcal{E}, \mathcal{B}(\mathcal{E}, o))] \\ & = & (\sum o \, : \, (\delta' \restriction L)(o) \cdot \mathcal{Q}(\mathcal{E}, ([\![S]\!](\dot{\sigma}_L \otimes b_H)|o) \restriction H)) \end{array}$$

where $\delta' \restriction L = [\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H) \restriction L$ is the distribution on observations; $\mathsf{E}_{\sigma \in \delta}[X(y)]$ is the expected value of expression X, which has free variable y, with respect to distribution δ; and $\mathcal{B}$ is the belief revision operator from §2.2.3.

Expected flow is useful in analyzing probabilistic programs. Consider a faulty password checker:

$$\begin{array}{ll} \mathit{FPWC} : & \textbf{if } p = g \textbf{ then } a := 1 \textbf{ else } a := 0; \\ & a := \neg a \ {}_{0.1}[\!] \ \textbf{skip} \end{array}$$

With probability 0.1, FPWC inverts the authentication flag. Can this program be expected to confound attackers—does FPWC leak less expected information than PWC? This question can be answered by comparing the expected flow from FPWC to the flow of PWC. Table 2.4 gives information flows from FPWC for experiments $\mathcal{E}_1^F$ and $\mathcal{E}_2^F$, which are identical to $\mathcal{E}_1$ and $\mathcal{E}_2$ from §2.3.1, except that they execute FPWC instead of PWC. Observations $(a \mapsto 0)$ and $(a \mapsto 1)$ correspond to an execution where the value of a is inverted. The flow for the outcomes resulting from these observations is negative, indicating that the program is giving the attacker misinformation. Note that, for both pairs of experiments in table 2.4, the expected flow of FPWC is less than the flow of PWC. We have confirmed that the random corruption of a makes it more difficult for the attacker to increase the accuracy of his belief.

Table 2.4: Leakage of PWC and $FPWC$

$\mathcal{E}$	o	$\mathcal{Q}(\mathcal{E}, \mathcal{B}(\mathcal{E}, o))$	$\mathcal{Q}_{\mathsf{E}}(\mathcal{E})$
$\mathcal{E}_1$	$(a \mapsto 1)$	0.0291	0.0291
	$(a \mapsto 0)$	impossible	
$\mathcal{E}_1^F$	$(a \mapsto 1)$	0.0258	0.0018
	$(a \mapsto 0)$	-0.2142	
$\mathcal{E}_2$	$(a \mapsto 1)$	impossible	5.6439
	$(a \mapsto 0)$	5.6439	
$\mathcal{E}_2^F$	$(a \mapsto 1)$	-3.1844	2.3421
	$(a \mapsto 0)$	2.9561	

Expected flow can be conservatively approximated by conditioning on a single distribution rather than conditioning on many observations. Conditioning δ on δ_L has the effect of making the low projection of δ identical to δ_L, while leaving the high projection of $\delta|\sigma_L$ unchanged for all σ_L:

$$\delta|\delta_L \quad \triangleq \quad \lambda\sigma \,.\, \frac{\delta(\sigma)}{(\delta \restriction L)(\sigma \restriction L)} \cdot \delta_L(\sigma \restriction L).$$

A bound on expected flow is then calculated as follows. Given experiment $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle$, let δ' be the distribution that results from the system executing S as in step 4 of the experiment protocol, i.e., $\delta' = [\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H)$. In the experiment protocol, an attacker would observe the low projection of a state from δ'. But suppose that the attacker instead observed the low projection of δ' itself. (This projection is the distribution over observations that the attacker would approach if he continued to repeat $\mathcal{E}$.) Let e_H be the postbelief that results from conditioning on this distribution, as in step 5 of the protocol: $e_H = (([\![S]\!](\dot{\sigma}_L \otimes b_H))|(\delta' \restriction L)) \restriction H$. Intuitively, e_H is the attacker's expected postbelief with respect to $\delta' \restriction L$. The amount of information flow from expected postbelief e_H then bounds the expected amount of information flow:

Theorem 2.6. *Let:*

$$\begin{aligned}
\mathcal{E} &= \langle S, b_H, \sigma_H, \sigma_L \rangle, \\
\delta' &= [\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H), \\
e_H &= (([\![S]\!](\dot{\sigma}_L \otimes b_H)) | (\delta' \upharpoonright L)) \upharpoonright H.
\end{aligned}$$

Then:

$$\mathcal{Q}_{\mathsf{E}}(\mathcal{E}) \quad \leq \quad \mathcal{Q}(\mathcal{E}, e_H).$$

Proof. In appendix 2.A. □

As an example, consider experiment $\mathcal{E}_2^F$. Calculating the attacker's expected postbelief e_H in this experiment yields $e_H = \langle 0.8601, 0.0699, 0.0699 \rangle$, using the postbelief notation from §2.3.3. Bound $\mathcal{Q}(\mathcal{E}, e_H)$ from theorem 2.6 is thus 6.4264 bits, which is indeed greater than expected flow $\mathcal{Q}_{\mathsf{E}}$ as calculated in table 2.4.

2.3.6 Expected Flow over All Experiments

Uncertainty-based metrics typically consider the expected information flow over all experiments, rather than the flow in a single experiment. An analysis, like ours, based on single experiments allows a more expressive language of security properties in which particular inputs or experiments can be considered. Moreover, our analysis can be extended to calculate expected flow over all experiments.

Rather than choosing particular high input states σ_H, the system may choose distribution δ_H over high states. A distribution over high inputs could be used, for example, to determine the expected flow of the password checker when users' choice of passwords can be described by a distribution. Distribution δ_H is sampled to produce the initial high input state. Taking the expectation in $\mathcal{Q}_{\mathsf{E}}$

with respect to both σ_H and o then yields the expected flow over all experiments for a given low input σ_L.

The expected flow over all experiments can be characterized using conditional mutual information (c.f. §2.3.4). Let H_{in} denote the distribution over high inputs, L_{in} over low inputs, and L_{out} over low outputs. For an experiment $\mathcal{E} = \langle S, b_H, \delta_H, \sigma_L \rangle$, distribution H_{in} is δ_H, distribution L_{in} is $\dot{\sigma}_L$, and distribution L_{out} is $\delta' \upharpoonright L$, where δ' is the output distribution from the experiment protocol. If system distribution δ_H is identical to attacker prebelief b_H (i.e., there is no error in the attacker's estimate of the probability distribution on high inputs), the expected flow over all experiments for a given low input is equal to the conditional mutual information between H_{in} and L_{out} given L_{in}:

Theorem 2.7. *Let $\mathcal{E} = \langle S, b_H, \delta_H, \sigma_L \rangle$, where $b_H = \delta_H$. Then:*

$$\mathcal{Q}_{\mathsf{E}}(\mathcal{E}) \quad = \quad \mathcal{I}(H_{in}, L_{out} | L_{in}).$$

Proof. In appendix 2.A. □

This theorem means that our metric for expected information flow agrees with uncertainty metrics (such as Clark et al. [24]) if attackers have beliefs that do not differ from reality—that is, if the attacker's belief is equal to the system's distribution on high inputs. This requirement is unsurprising, because uncertainty metrics do not distinguish between beliefs and reality.

The attacker may also choose distribution δ_L over low states. This extension increases the expressive power of the experiment model—for example, the attacker can use δ_L to express a randomized guessing strategy. His distribution might also be a function of his belief; we do not address such attacker strategies here.

2.3.7 Maximum Information Flow

System designers are likely to want to limit the maximum possible information flow. We characterize the maximum amount of information flow that program S can cause in a single outcome as the maximum amount of flow from any outcome of any experiment $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle$ on S:

$$\mathcal{Q}_{\max}(S) \quad \triangleq \quad \max\{\mathcal{Q}(\mathcal{E}, b'_H) \mid \mathcal{E}, b'_H \in \mathcal{B}(\mathcal{E})\}.$$

Consider applying $\mathcal{Q}_{\max}$ to PWC. Assume that b_H is a uniform distribution, representing a lack of belief for any particular password, over k-bit passwords. If the attacker guesses correctly, the maximum leakage is k bits according to $\mathcal{Q}_{\max}$. But if the attacker guesses incorrectly, PWC can leak at most $k - \lg(2^k - 1)$ bits in an outcome; for $k > 12$ this is less than 0.0001 bits.

Uncertainty metrics typically declare that the maximum possible information flow is $\lg |\mathbf{State}_H|$; this is the number of bits necessary to store the high state. This was true for the example of k-bit passwords above. However, as experiment $\mathcal{E}_2$ from §2.3.1 shows, this declaration is valid only if the attacker's prebelief is no more inaccurate than the uniform distribution. Thus uncertainty metrics make an implicit restriction on attacker beliefs that our accuracy metric does not.

2.3.8 Repeated Experiments

Nothing precludes performing a series of experiments. The most interesting case has the attacker return to step 2b of the experiment protocol in figure 2.2 after updating his belief in step 5—that is, the system keeps the high input to the program constant, and the attacker is allowed to check new low inputs based on the results of previous experiments.

Table 2.5: Repeated experiments on PWC

		Repetition	
		1	2
b_H :	A	0.98	0
	B	0.01	0.5
	C	0.01	0.5
$\sigma_L(g)$		A	B
$o(a)$		0	0
b'_H :	A	0	0
	B	0.5	0
	C	0.5	1
$\mathcal{Q}(\mathcal{E}, b'_H)$		5.6439	1.0

Suppose that experiment $\mathcal{E}_2$ from §2.3.1 is conducted and repeated with $\sigma_L = (g \mapsto B)$. Then the attacker's belief about the password evolves as shown in table 2.5. Summing the information flow for each experiment yields a total information flow of 6.6439. This total corresponds to what $\mathcal{Q}$ would calculate for a single experiment, if that experiment changed prebelief b_H to postbelief b'_{H2}, where b'_{H2} is the attacker's final postbelief in table 2.5:

$$\begin{aligned} D(b_H \rightarrow \dot{\sigma}_H) - D(b'_{H2} \rightarrow \dot{\sigma}_H) &= 6.6439 - 0 \\ &= 6.6439 \end{aligned}$$

This example suggests that, given a series of experiments in which the postbelief from one experiment becomes the prebelief to the next, the final postbelief contains all the information learned during the series. Let $\mathcal{E}_i = \langle S, b_{H_i}, \sigma_H, \sigma_{L_i} \rangle$ be the i^{th} experiment in the series, and let b'_{H_i} be the outcome from $\mathcal{E}_i$. Let prebelief b_{H_i} in experiment $\mathcal{E}_i$ be chosen as postbelief $b'_{H_{i-1}}$ from experiment $\mathcal{E}_{i-1}$. Let b_{H_1} be the attacker's prebelief for the entire series. Let n be the length of the series. The following theorem states that the final postbelief does contain all the information:

Theorem 2.8. $D(b_{H_1} \rightarrow \dot{\sigma}_H) - D(b'_{H_n} \rightarrow \dot{\sigma}_H) = (\sum i : 1 \leq i \leq n : \mathcal{Q}(\mathcal{E}_i, b'_{H_i}))$.

Proof. Immediate by the definition of $\mathcal{Q}$ and arithmetic. □

Consequently, our experiment model enables compositional reasoning about series of attacks.

2.3.9 Number of Experiments

Attackers conduct experiments to refine their beliefs. This suggests another quantification of the security of a program: the number of experiments required for an attacker to refine his belief to within some distance of reality. For simplicity, assume that program S is deterministic,[6] such that only one observation is possible from an experiment. Then belief revision $\mathcal{B}$ (from §2.2.3) can be used as a function from experiments to postbeliefs. Let $\mathcal{A}$: **Belief** $\rightarrow$ **State**$_L$ be the attacker's *strategy* for choosing low inputs based on his beliefs. Define the i^{th} iteration of $\mathcal{B}$ as $\mathcal{B}^i$:

$$\begin{array}{rcl} \mathcal{B}^i(S, b_H, \sigma_H, \mathcal{A}) & \triangleq & \mathcal{B}(S, b'_H, \sigma_H, \mathcal{A}(b'_H)), \\ & & \text{where } b'_H = \mathcal{B}^{i-1}(S, b_H, \sigma_H, \mathcal{A}); \\ \mathcal{B}^1(S, b_H, \sigma_H, \mathcal{A}) & \triangleq & \mathcal{B}(S, b_H, \sigma_H, \mathcal{A}(b_H)). \end{array}$$

Then the number of experiments $\mathcal{N}$ needed to achieve a postbelief within distance ϵ of reality is:

$$\mathcal{N}(S, b_H, \sigma_H, \mathcal{A}) \triangleq \min\{i \mid D(\mathcal{B}^i(S, b_H, \sigma_H, \mathcal{A}) \rightarrow \sigma_H) \leq \epsilon\}.$$

As discussed in §2.3.2, when an attacker's belief is k bits distant from reality, the probability he ascribes to the correct high state is $1/2^k$. If the attacker

[6]If program S is probabilistic, $\mathcal{B}(\mathcal{E})$ could instead be defined as a random variable giving the probability with which the attacker holds a postbelief. This would allow the definition of the expected number of experiments to achieve a distance from reality.

$$\begin{array}{rcl}
[\![\textbf{skip}]\!]\sigma & = & \dot{\sigma} \\
[\![v := E]\!]\sigma & = & \dot{\sigma}[v \mapsto E] \\
[\![S_1; S_2]\!]\sigma & = & [\![S_2]\!]^*([\![S_1]\!]\sigma) \\
[\![\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2]\!]\sigma & = & \text{if } [\![B]\!]\sigma \text{ then } [\![S_1]\!]\sigma \text{ else } [\![S_2]\!]\sigma \\
[\![\textbf{while } B \textbf{ do } S]\!] & = & \text{fix}(\lambda d : \textbf{State} \rightarrow \textbf{Dist} \,. \\
& & \quad \lambda\sigma \,.\, \text{if } [\![B]\!]\sigma \text{ then } d^*([\![S]\!]\sigma) \text{ else } \dot{\sigma}) \\
[\![S_1 \ {}_p[\!] \ S_2]\!]\sigma & = & p \cdot [\![S_1]\!]\sigma + (1-p) \cdot [\![S_2]\!]\sigma
\end{array}$$

Figure 2.4: State semantics of programs

were to guess a high state by sampling from his belief, he would therefore guess correctly with probability $1/2^{\epsilon}$ after $\mathcal{N}$ experiments.

Sometimes an attacker needs only to reach a belief that is close to reality. For example, if the high state is a Cartesian coordinate, the attacker might need only to bound the coordinate within some Cartesian distance. Let $ball(\sigma_H)$ be all the high states within distance γ of σ_H according to a distance metric M on $\textbf{State}_H$:

$$ball(\sigma_H) \ \triangleq \ \{\sigma'_H \mid M(\sigma'_H \twoheadrightarrow \sigma_H) \leq \gamma\}.$$

Then the number of experiments needed to achieve some distance ϵ from some ball γ around reality is:

$$\mathcal{N}(S, b_H, \sigma_H, \mathcal{A}) \ \triangleq \ \min\{i \mid \sigma'_H \in ball(\sigma_H) \ \wedge \ D(\mathcal{B}^i(S, b_H, \sigma_H, \mathcal{A}) \twoheadrightarrow \sigma'_H) \leq \epsilon\}.$$

2.4 Language Semantics

The last technical piece we require is a semantics $[\![S]\!]$ in which programs denote functions that map distributions to distributions. Here we build such a semantics in two stages. First, we build a simpler semantics that maps states to distributions. Second, we lift that semantics so that it operates on distributions.

Our first task then is to define the semantics $[\![S]\!] : \textbf{State} \rightarrow \textbf{Dist}$. That semantics is given in figure 2.4. We assume a semantics $[\![E]\!] : \textbf{State} \rightarrow \textbf{Val}$ that gives meaning to expressions, and a semantics $[\![B]\!] : \textbf{State} \to \textbf{Bool}$ that gives meaning to Boolean expressions.

The statements **skip** and **if** have essentially the same denotations as in the standard deterministic case. State update $\sigma[v \mapsto V]$, where $V \in \textbf{Val}$, changes the value of v to V in σ. The distribution update $\delta[v \mapsto E]$ in the denotation of assignment represents the result of substituting the meaning of E for v in all the states of δ:

$$\delta[v \mapsto E] \quad \triangleq \quad \lambda\sigma\,.\,((\textstyle\sum \sigma' \,:\, \sigma'[v \mapsto [\![E]\!]\sigma'] = \sigma \,:\, \delta(\sigma'))).$$

The semantics of **while** and sequential composition $S_1; S_2$ use lifting operator *, which lifts function $d : \textbf{State} \to \textbf{Dist}$ to function $d^* : \textbf{Dist} \to \textbf{Dist}$, as suggested by §2.1.2:

$$\begin{aligned} d^* &\triangleq \lambda\delta\,.\,(\textstyle\sum \sigma \,:\, \delta(\sigma) \cdot d(\sigma)) \\ &= \lambda\delta\,.\,\lambda\sigma\,.\,(\textstyle\sum \sigma' \,:\, \delta(\sigma') \cdot d(\sigma')(\sigma)), \end{aligned}$$

where the equality follows from η-reduction, and $\cdot$ and $+$ are used as pointwise operators:

$$\begin{aligned} p \cdot \delta &\triangleq \lambda\sigma\,.\,p \cdot \delta(\sigma), \\ \delta_1 + \delta_2 &\triangleq \lambda\sigma\,.\,\delta_1(\sigma) + \delta_2(\sigma). \end{aligned}$$

Lifted d^* is thus the expected value (which is a distribution) of d with respect to distribution δ.

To ensure that the fixed point for **while** exists, we must verify that **Dist** is a complete partial order with a bottom element and that $[\![\cdot]\!]$ is continuous. We omit the proof here, as it is a consequence of a theorem proved by Kozen [66]. But we note that a key step is to strengthen the definition of **Dist** from §2.1.1 to be $\{\delta \mid \delta \in \textbf{State} \to [0,1] \ \wedge\ \|\delta\| \leq 1\}$. This makes distributions correspond to subprobability measures, and it is easy to check that the semantics produces subprobability measures as output. The bottom element is then $\lambda\sigma\,.\,0$, and the ordering relation on distributions is pointwise. Note that the definition of **Belief** from §2.1.4 remains unchanged, since it did not depend on **Dist**. Thus beliefs still correspond to probability measures. Anywhere that the result of the program semantics must be upgraded to a belief (i.e., from a subprobability to a probability), we rely on the technique of §2.2.4 to handle nontermination. The most important occurrence of this is in step 5 of the experiment protocol in figure 2.2.

The final program construct is probabilistic choice, $S_1 \ {}_p[\!]\ S_2$, where $0 \leq p \leq 1$. The semantics multiplies the probability of choosing a side S_i with the frequency that S_i produces a particular output state σ'. Since the same state σ' might actually be produced by both sides of the choice, the frequency of its occurrence is the sum of the frequency from either side: $p \cdot ([\![S_1]\!]\sigma)(\sigma') + (1-p) \cdot ([\![S_2]\!]\sigma)(\sigma')$, which can be simplified to the formula in figure 2.4.

To lift the semantics in figure 2.4 and define $[\![S]\!] : \mathbf{Dist} \to \mathbf{Dist}$, we again employ lifting operator *:

$$\begin{aligned} [\![S]\!]\delta &\triangleq [\![S]\!]^*\delta \\ &= \lambda\sigma \,.\, (\textstyle\sum \sigma' : \delta(\sigma') \cdot ([\![S]\!]\sigma')(\sigma)). \end{aligned}$$

Interpreting this definition, note there are many states σ' in which S could begin execution, and all of them could potentially terminate in state σ. So to compute $([\![S]\!]\delta)(\sigma)$, we take a weighted average over all input states σ'. The weights are $\delta(\sigma')$, which describes how likely σ' is to be used as the input state. With σ' as input, S terminates in state σ with frequency $([\![S]\!]\sigma')(\sigma)$.

Applying this definition to the semantics in figure 2.4 yields $[\![S]\!]\delta$, shown in figure 2.5. This lifted semantics corresponds directly to a semantics given by Kozen [66], which interprets programs as continuous linear operators on probability measures. Our semantics uses an extension of the distribution con-

$$\begin{array}{rcl}
[\![\mathbf{skip}]\!]\delta & = & \delta \\
[\![v := E]\!]\delta & = & \delta[v \mapsto E] \\
[\![S_1; S_2]\!]\delta & = & [\![S_2]\!]([\![S_1]\!]\delta) \\
[\![\mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2]\!]\delta & = & [\![S_1]\!](\delta \mid B) + [\![S_2]\!](\delta \mid \neg B) \\
[\![\mathbf{while}\ B\ \mathbf{do}\ S]\!] & = & \mathrm{fix}(\lambda d : \mathbf{Dist} \to \mathbf{Dist}\,.\,\lambda\delta\,.\,d([\![S]\!](\delta \mid B)) + (\delta \mid \neg B)) \\
[\![S_1\ {}_p[\!]\ S_2]\!]\delta & = & [\![S_1]\!]p \cdot \delta + [\![S_2]\!](1-p) \cdot \delta
\end{array}$$

Figure 2.5: Distribution semantics of programs

ditioning operator | to Boolean expressions. Whereas distribution conditioning produces a normalized distribution, Boolean expression conditioning produces an unnormalized distribution:

$$\delta|B \quad \triangleq \quad \lambda\sigma\,.\,\text{if } [\![B]\!]\sigma \text{ then } \delta(\sigma) \text{ else } 0.$$

By producing unnormalized distributions as part of the meaning of **if** and **while** statements, we track the frequency with which each branch of the statement is chosen.

2.5 Insider Choice

The experiment protocol in §2.2 involved two agents, the attacker and the system. Consider a third agent called the *insider*, whose goal is to help the attacker learn secret information. The insider and attacker might initially communicate to establish a strategy to achieve this goal. Once execution begins, the insider cannot directly communicate with the attacker, but the insider can observe the entire program state and can influence execution.

The insider's ability to influence execution is modeled by a new programming language construct, *insider choice*, denoted $S_1\ [\!]\ S_2$:

$$S \ ::= \ \ldots \mid S_1\ [\!]\ S_2$$

The insider, rather than the system, is the entity who executes this kind of choice. The insider chooses either S_1 or S_2 and execution continues with the chosen program.

As an example of insider choice, consider program $L1$:

$$\begin{aligned} L1: \quad & h := h \mathsf{~mod~} 2; \\ & l := 0 \ [] \ l := 1 \end{aligned}$$

The second line of $L1$ allows the insider to choose between two values for variable l. Since the insider is allowed to observe the high component of the state, he can observe the parity of h and choose to set l equal to it, thus leaking the parity of h.

The insider in this example made a deterministic choice. More generally, insiders may also make probabilistic choices. For example, an insider could flip a fair coin then choose the left side on heads or the right side on tails. This can be seen as an extension of probabilistic choice, in which the probability is a function of the program state rather than just a constant. Thus insider choice can model the behavior of probabilistic programs that are not influenced by an insider.

2.5.1 Insider Functions

Formally, an insider is a function $I \in \textbf{Insider}$, where

$$\textbf{Insider} \quad \triangleq \quad \textbf{State} \to [0..1].$$

$I(\sigma)$ is the probability with which the left-hand side of the insider choice is taken. For example, insider function I_{L1} leaks the value of h in program $L1$ with probability 0.99:

$$I_{L1}(\sigma) \quad = \quad \text{if } \sigma(h) = 0 \text{ then } 0.99 \text{ else } 0.01$$

In a program with multiple syntactic occurrences of insider choice, a single insider function can encode different probabilities for each occurrence if the program state encodes the program counter.

Moreover, if the program state is sufficiently rich, insider functions can model a range of insider capabilities. For example, suppose the operational semantics guarantees that for every variable x, the previous value of x (i.e., the value that was assigned to it before its current value was assigned) is preserved in variable $\grave{x}$. Then insider functions can make decisions based on past state by reading those previous values.[7] In the following program, the insider leaks the initial parity of h:

$$\begin{aligned} LP: \quad & h := h \ \mathsf{mod}\ 2; \\ & h := 0; \\ & l := 0 \ \,[\!]\ \, l := 1 \end{aligned}$$

The insider function that accomplishes this is

$$I_{LP}(\sigma) \quad = \quad \text{if } \sigma(\grave{h}) = 0 \text{ then } 1 \text{ else } 0.$$

Note that without access to variable $\grave{h}$, the insider is unable to leak the initial parity of h because this information is removed from the state when h is assigned the value 0.

Insiders with limited computational resources can be modeled by further restricting **Insider**. For example, suppose that insiders are allowed only polynomial time to make a choice. Then insider functions could be replaced by polynomially time-bounded Turing machines, where the input to the machine is the input σ to the insider function, and the output of the machine is used as the output of the insider function.

[7]This mechanism is similar to *history variables* [1]. Likewise, insiders who can predict the future values of variables could be modeled by a mechanism similar to *prophecy variables* [1].

$$
\begin{array}{rcl}
[\![\textbf{skip}]\!]_I\sigma & = & \dot{\sigma} \\
[\![v := E]\!]_I\sigma & = & \dot{\sigma}[v \mapsto E] \\
[\![S_1; S_2]\!]_I\sigma & = & ([\![S_2]\!]_I)^*([\![S_1]\!]_I\sigma) \\
[\![\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2]\!]_I\sigma & = & \text{if } [\![B]\!]\sigma \text{ then } [\![S_1]\!]_I\sigma \text{ else } [\![S_2]\!]_I\sigma \\
[\![\textbf{while } B \textbf{ do } S]\!]_I & = & \text{fix}(\lambda d : \textbf{State} \to \textbf{Dist}\,. \\
 & & \quad \lambda\sigma\,.\,\text{if } [\![B]\!]\sigma \text{ then } d^*([\![S]\!]_I\sigma) \text{ else } \dot{\sigma}) \\
[\![S_1 \;{}_p[\!] \; S_2]\!]_I\sigma & = & p \cdot [\![S_1]\!]_I\sigma + (1-p) \cdot [\![S_2]\!]_I\sigma \\
[\![S_1 \;\; [\!] \;\; S_2]\!]_I\sigma & = & I(\sigma) \cdot [\![S_1]\!]_I\sigma + (1 - I(\sigma)) \cdot [\![S_2]\!]_I\sigma
\end{array}
$$

Figure 2.6: State semantics of programs with insider

$$
\begin{array}{rcl}
[\![\textbf{skip}]\!]_I\delta & = & \delta \\
[\![v := E]\!]_I\delta & = & \delta[v \mapsto E] \\
[\![S_1; S_2]\!]_I\delta & = & [\![S_2]\!]_I([\![S_1]\!]_I\delta) \\
[\![\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2]\!]_I\delta & = & [\![S_1]\!]_I(\delta \,|\, B) + [\![S_2]\!]_I(\delta \,|\, \neg B) \\
[\![\textbf{while } B \textbf{ do } S]\!]_I & = & \text{fix}(\lambda d : \textbf{Dist} \to \textbf{Dist}\,. \\
 & & \quad \lambda\delta\,.\,d([\![S]\!]_I(\delta \,|\, B)) + (\delta \,|\, \neg B)) \\
[\![S_1 \;{}_p[\!] \; S_2]\!]_I\delta & = & [\![S_1]\!]_I\, p \cdot \delta + [\![S_2]\!]_I(1-p) \cdot \delta \\
[\![S_1 \;\; [\!] \;\; S_2]\!]_I\delta & = & [\![S_1]\!]_I I(\delta) + [\![S_2]\!]_I \overline{I}(\delta)
\end{array}
$$

Figure 2.7: Distribution semantics of programs with insider

2.5.2 Semantics and Experiments

Formal semantics $[\![S]\!] : \textbf{Insider} \to \textbf{State} \to \textbf{Dist}$ is given in figure 2.6. The only place in the semantics that the insider function is used is in the semantics of $S_1 \;\, [\!] \;\, S_2$, and the semantics never modifies the insider function. Because of this second-class nature of insider functions, and for improved readability, we use a subscript notation for the insider function I in semantics $[\![S]\!]_I$. We can lift the semantics to operate on distributions as shown in figure 2.7. The lifted insider function is defined as follows:

$$
\begin{array}{rcl}
I(\delta) & \triangleq & \lambda\sigma\,.\,I(\sigma) \cdot \delta(\sigma), \\
\overline{I}(\delta) & \triangleq & \lambda\sigma\,.\,(1 - I(\sigma)) \cdot \delta(\sigma).
\end{array}
$$

The experiment protocol in §2.2.1 can be extended to include insiders, as shown in figure 2.8. Note that the attacker uses insider function I when con-

An experiment $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L, I \rangle$ is conducted as follows.

1. The attacker chooses a prebelief b_H about the high state.
2. (a) The system picks a high state σ_H.
   (b) The attacker picks a low state σ_L.
3. The attacker predicts the output distribution: $\delta'_A = [\![S]\!]_I(\dot{\sigma}_L \otimes b_H)$.
4. The system and insider execute the program S, which produces a state $\sigma' \in \delta'$ as output, where $\delta' = [\![S]\!]_I(\dot{\sigma}_L \otimes \dot{\sigma}_H)$. The attacker observes the low projection of the output state: $o = \sigma' \upharpoonright L$.
5. The attacker infers a postbelief: $b'_H = (\delta'_A | o) \upharpoonright H$.

Figure 2.8: Experiment protocol with insider

ducting the thought-experiment. This function thus encodes choices that the insider and attacker have agreed upon in advance.

2.5.3 Security Conditions

Observational determinism [85,102,130] is a security condition for nondeterministic systems that generalizes noninterference [46]. We can state a probabilistic generalization of observational determinism that is applicable to our insider model: a program S satisfies observational determinism exactly when S behaves as a function from a low input state to a low output distribution, for any insider and high input. Let the set of programs satisfying observational determinism be denoted **ObsDet**, which is defined as follows:

$$\textbf{ObsDet} \quad \triangleq \quad \{S \mid \forall I \,.\, \forall \sigma_L \,.\, \exists \delta_L \,.\, \forall \sigma_H \,.\, [\![S]\!]_I(\dot{\sigma}_L \otimes \dot{\sigma}_H) \upharpoonright L = \delta_L\}.$$

Observational determinism is equivalent to zero information flow in the insider model—that is, a program S satisfies observational determinism exactly when all experiments over S leak exactly 0 bits of information:

Theorem 2.9. $S \in \mathbf{ObsDet} \;\equiv\; \forall \mathcal{E}, b'_H \in \mathcal{B}(\mathcal{E}) \,.\, \mathcal{Q}(\mathcal{E}, b'_H) = 0.$

Proof. In appendix 2.A. $\square$

Theorem 2.9 suggests that observational determinism is the absolute security condition for nondeterministic systems. On the other hand, the theorem also shows that observational determinism is too strong to be useful with programs that require information flow, such as PWC.

Other nondeterministic security conditions, such as generalized noninterference (GNI) [81], are already known to allow leakage of information [119]. Our model of insider choice allows this leakage to be quantified: a program S satisfies GNI when S behaves as a relation on a low input state and low output distributions, for any insider and high input:

$$\mathbf{GNI} \quad \triangleq \quad \{S \mid \forall \sigma_L \,.\, \exists \Delta_L \,.\, \forall \sigma_H \,.\, \bigcup_I ([\![S]\!]_I(\dot{\sigma}_L \otimes \dot{\sigma}_H) \upharpoonright L) = \Delta_L\}.$$

Consider program LH, which can be shown to be in **GNI**:

$$LH: \quad l := h \;\;[\!]\;\; (l := 0 \;\;[\!]\;\; l := 1)$$

Using insider function $I_{LH}(\sigma) = 1$, this program always leaks the value of h. Unless the attacker already has a perfectly accurate belief about h, this is a positive (and non-zero) amount of leakage. So even though the program is secure according to **GNI**, an insider can refine the program to be insecure. This weakness is known as the *refinement paradox* [102]. Insiders therefore introduce a kind of nondeterminism that is not secure under refinement.

2.6 Related Work

Quantification of information flow. The first published connection between information theory and information flow is by Denning [35], who uses entropy to calculate the leakage of a few assignment and conditional statements.

Backes, Köpf, and Rybalchenko [11] construct an automated static analysis for computing the quantity of information flow in simple imperative programs. Their analysis assumes a uniform distribution on high inputs, computes a high equivalence relation on low observable outputs, then counts the number of high inputs in each equivalence class. This count yields a probability distribution that can be used to compute several entropy-based metrics of information flow.

Smith [108] argues that the function used to quantify uncertainty should depend on the attack model. For some programs, the expectation taken as part of the formula for mutual information masks the fact that certain executions leak a large amount of information, thus making it easy for the attacker to guess the remaining secret information. Our framework in part addresses this problem by allowing quantification of information flow both for single experiments and in expectation over all experiments.

McCamant and Ernst [80] implement an automated hybrid analysis for quantification of information flow in Linux/x86 binaries. Their analysis computes a conservative upper bound on the amount of information that can be leaked by the particular execution the dynamic part of the analysis observes. But the analysis cannot bound the quantity of flow for executions it does not observe. The quantity measured by the analysis is an upper bound on *channel capacity*, which is the maximum amount, over any probability distribution on inputs, of mutual information between secret inputs and public outputs.

Köpf and Basin [65] quantify the resistance of a deterministic system against sequences of attacks, where *resistance* is a function from the number of attacks performed to the expected remaining uncertainty of the attacker. Their definitions can quantify uncertainty with several variants of entropy. They give an automated, heuristic analysis that approximates resistance.

Clark, Hunt, and Malacaria [24] develop a static analysis that bounds the amount of information leaked by a **while**-program. Their metric for information leakage is based on conditional entropy. The analysis comprises a dataflow analysis, which computes a use-def graph, and syntax-directed inference rules, which calculate leakage bounds. These authors also investigate other leakage metrics, settling on conditional mutual information as an appropriate metric for quantification of flow in probabilistic languages [23]; they do not consider relative entropy. Mutual information is always at least 0, so unlike relative entropy it cannot represent misinformation. As noted in §2.3.4, this uncertainty-based definition requires a strong admissibility restriction: the attacker's prebelief must be the same distribution from which the system generates the high input. Malacaria [77] extends this line of work by classifying the rate of leakage of loops. His basic definition of amount of leakage is equivalent to [24], so it is an instance of our own definition, as shown in §2.3.4. For the same reason, Malacaria's model is no more precise than our own model. Rate of leakage could be defined in our own model, like the other statistics in §2.3.

Backes [10] quantifies information flow for reactive systems, which execute cryptographic protocols, as the maximum distance between the low user's views of a protocol run for any two high behaviors, where a *view* is a probability distribution on the traces observed by the user. The distance metric is left abstract, hence not instantiated by any information-theoretic definition.

Di Pierro, Hankin, and Wiklicky [38] relax noninterference to *approximate noninterference*, where "approximate" denotes similarity of two processes in a process algebra; similarity is quantified using the supremum norm of the difference between the probability distributions that the processes create on memory. This quantity can be interpreted as a probability on an attacker's ability to distinguish the two processes using a finite number of tests. This work also builds an abstract interpretation that allows approximation of the confinement of a process. Subsequent work [39] generalizes from process algebras to probabilistic transition systems.

Lowe [76] defines the *information flow quantity* of a process with two users H and L to be the number of behaviors of H that L can distinguish. When there are n such distinguishable behaviors, H can use them to transmit $\lg n$ bits to L.

Weber [123] defines *n-limited security*, which allows declassification at a rate that depends, in part, on the size n of a buffer shared by the high and low projections of a state.

Millen [88], using deterministic state machines, proves that a system satisfies noninterference exactly when the mutual information between certain inputs and outputs is zero. He also proposes mutual information as a metric for information flow, but he does not show how to compute the amount of flow for programs.

Database privacy. Evfimievski, Gehrke, and Srikant [42] quantify *privacy breaches* in data mining. In their framework, randomized operators are applied to confidential data before the data is released. A privacy breach occurs when release of the randomized data causes a large change in an attacker's probability distribution on a property of the confidential data. They use Bayesian reasoning, based on observation of randomized data, to update the attacker's distribution. Their distributions are similar to our beliefs, but have the same strong admissibility restriction as Clark et al. [24] (c.f. §2.3.4). They also show that relative entropy can be used to bound the maximum privacy breach for a randomized operator.

Anonymity protocols. Chatzikokolakis et al. [21] analyze the degree of anonymity provided by anonymity protocols. They model protocols as channels, and they quantify the loss of anonymity introduced by a protocol as the information-theoretic capacity of the channel.

Noninterference. The flow model (FM) is a security property proposed by McLean [84] and later given a quantitative formalization by Gray [49], who called it the Applied Flow Model. The FM stipulates that the probability of a low output may depend on previous low outputs, but not on previous high outputs. Gray formalizes this in the context of probabilistic state machines, and he relates noninterference to the maximum rate of flow between high and low.

Browne [19] develops a novel application of the Turing test: a system passes Browne's Turing test exactly when for all finite lengths of time, the information flow over that time is zero.

Volpano [118] gives a type system that can be used to establish the security of password checking and one-way functions such as MD5 and SHA1. Noninterference does not usually allow such functions to be typed, so this type system is an improvement over previous type systems. However, the type system does not allow a general analysis of quantitative information flow.

Volpano and Smith [120] give a type system that enforces *relative secrecy*, which enforces that well-typed programs cannot leak confidential data in polynomial time.

Nondeterminism. Wittbold and Johnson [127] introduce *nondeducibility on strategies*, an extension of Sutherland's *nondeducibility* [113]. Wittbold and Johnson observe that if a program is run multiple times and feedback between runs is allowed, information can be leaked by coding schemes across multiple runs. A system that is nondeducible on strategies has no noiseless communication channels between high input and low output, even in the presence of feedback. Our insider framework can quantify the leakage due to strategies that are encodable as insider functions.

Halpern and Tuttle [53] introduce a framework for reasoning about knowledge and probability based on three kinds of adversaries: adversaries who make nondeterministic choices, adversaries who represent the knowledge of the opponent, and adversaries who control timing. Our insiders can be seen as an instantiation of this framework. The insider choice and insider function constitute an adversary who makes nondeterministic choices, and each of the models of the insider's power in §2.5.1 correspond to an adversary representing the knowledge of the opponent. Gray and Syverson [50] apply the Halpern-Tuttle framework to reason about qualitative security of probabilistic systems. They relate their security condition to probabilistic noninterference [49] and information theory. Halpern and O'Neill [51] construct a framework for reasoning about secrecy that generalizes many previous results on qualitative and probabilistic, but not quantitative, security. Their framework, like ours, uses subjective probability distributions.

McIver and Morgan [82] calculate the channel capacity of a program using conditional entropy. They add *demonic nondeterminism* as well as probabilistic choice to the language of **while**-programs, and they show that whether a program is perfectly secure (i.e., leaks 0 bits) is determined by the behavior of its deterministic refinements. They also consider restricting the observational power of the demon making the nondeterministic choices.

2.7 Summary

This chapter presents a model for incorporating attacker beliefs into analysis of quantitative information flow. Our theory reveals that uncertainty, the traditional metric for information flow, is inadequate. Information flows when an attacker's belief becomes more accurate, but an uncertainty metric can mistakenly report a flow of zero or less. Inversely, misinformation flows when an attacker's belief becomes less accurate, but an uncertainty metric can mistakenly report a positive information flow. Hence, in the presence of beliefs, accuracy is the correct metric for information flow.

We have shown how to use an accuracy metric to calculate exact, expected, and maximum information flow; other statistics of information flow, such as variance, median, and rate, could be defined in the same way. We have demonstrated that our metric generalizes uncertainty metrics. Our formal model of experiments enables precise, compositional reasoning about attackers' actions and beliefs. We have instantiated this model with a probabilistic semantics and have shown that probabilistic choice is essential to producing misinformation. We have also extended the model to enable analysis of information flow caused by insiders who collude with attackers.

2.A Appendix: Proofs

Theorem 2.1. $\mathcal{B}(\mathcal{E}, o)(\sigma_H) = BI(\mathcal{E}, o)$.

Proof.

$$BI(\mathcal{E}, o)$$

$=$ ⟨ Definition of BI ⟩

$$\frac{b_H(\sigma_H) \cdot (\llbracket S \rrbracket(\dot{\sigma}_L \otimes \dot{\sigma}_H) \restriction L)(o)}{(\sum \sigma'_H \,:\, b_H(\sigma'_H) \cdot (\llbracket S \rrbracket(\dot{\sigma}_L \otimes \dot{\sigma}'_H) \restriction L)(o))}$$

$=$ ⟨ Definition of $\delta \restriction L$, apply distribution to o ⟩

$$\frac{b_H(\sigma_H) \cdot ((\sum \sigma \,:\, \sigma \restriction L = o \,:\, (\llbracket S \rrbracket(\dot{\sigma}_L \otimes \dot{\sigma}_H)(\sigma)))}{(\sum \sigma'_H \,:\, b_H(\sigma'_H) \cdot ((\sum \sigma \,:\, \sigma \restriction L = o \,:\, (\llbracket S \rrbracket(\dot{\sigma}_L \otimes \dot{\sigma}'_H)(\sigma))))}$$

$=$ ⟨ Lemma 2.1 (below) ⟩

$$\frac{b_H(\sigma_H) \cdot ((\sum \sigma \,:\, \sigma \restriction L = o \,:\, (\llbracket S \rrbracket(\dot{\sigma}_L \otimes \dot{\sigma}_H)(\sigma)))}{(\sum \sigma' \,:\, \sigma' \restriction L = o \,:\, \llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H)(\sigma'))}$$

$=$ ⟨ Distributivity, one-point rule ⟩

$$\frac{(\sum \sigma \,:\, \sigma \restriction L = o \;\wedge\; \sigma \restriction H = \sigma_H \,:\, (\sum \sigma'_H \,:\, b_H(\sigma_H) \cdot \llbracket S \rrbracket(\dot{\sigma}_L \otimes \dot{\sigma}_H)(\sigma)))}{(\sum \sigma' \,:\, \sigma' \restriction L = o \,:\, \llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H)(\sigma'))}$$

$=$ ⟨ Lemma 2.1 (below) ⟩

$$\frac{(\sum \sigma \,:\, \sigma \restriction L = o \;\wedge\; \sigma \restriction H = \sigma_H \,:\, \llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H)(\sigma))}{(\sum \sigma' \,:\, \sigma' \restriction L = o \,:\, \llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H)(\sigma'))}$$

$=$ ⟨ Distributivity ⟩

$$(\sum \sigma \,:\, \sigma \restriction L = o \;\wedge\; \sigma \restriction H = \sigma_H \,:\, \frac{\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H)(\sigma)}{(\sum \sigma' \,:\, \sigma' \restriction L = o \,:\, \llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H)(\sigma'))})$$

$=$ ⟨ Definition of $\delta \restriction L$ ⟩

$$(\sum \sigma \,:\, \sigma \restriction H = \sigma_H \,:\, ((\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H))|o)(\sigma))$$

$=$ ⟨ Definition of $\delta \restriction H$, applying distribution to σ_H ⟩

$$(((\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H))|o) \restriction H)(\sigma_H)$$

$=\quad \langle$ Definition of $\mathcal{B}(\mathcal{E}, o)$ $\rangle$

$\quad\mathcal{B}(\mathcal{E}, o)(\sigma_H)$

$\square$

Lemma 2.1. *Let $\sigma \upharpoonright L = o$. Then:*

$$[\![S]\!](\dot{\sigma}_L \otimes b_H)(\sigma) \;=\; (\textstyle\sum \sigma_H \,:\, b_H(\sigma_H) \cdot [\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H)(\sigma)).$$

Proof.

$\quad[\![S]\!](\dot{\sigma}_L \otimes b_H)(\sigma)$

$=\quad \langle$ Definition of $[\![S]\!]\delta$ $\rangle$

$\quad(\sum \sigma' \,:\, (\dot{\sigma}_L \otimes b_H)(\sigma') \cdot ([\![S]\!]\sigma')(\sigma))$

$=\quad \langle$ Definition of point mass $\rangle$

$\quad(\sum \sigma' \,:\, \sigma' \upharpoonright L = \sigma_L \,:\, b_H(\sigma' \upharpoonright H) \cdot ([\![S]\!]\sigma')(\sigma))$

$=\quad \langle$ Let $\sigma = \sigma_L \cup \sigma_H$, nesting, one-point rule $\rangle$

$\quad(\sum \sigma_H \,:\, b_H(\sigma_H) \cdot [\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H)(\sigma))$

$\square$

Theorem 2.2. $\mathcal{Q}(\mathcal{E}, b'_H) = \mathcal{I}_{\delta_A}(o) - \mathcal{I}_{\delta_S}(o)$.

Proof.

$\quad\mathcal{Q}(\mathcal{E}, b'_H)$

$=\quad \langle$ Definition of $\mathcal{Q}$ $\rangle$

$\quad D(b_H \rightarrow \dot{\sigma}_H) - D(b'_H \rightarrow \dot{\sigma}_H)$

$=\quad \langle$ Definitions of D and point mass $\rangle$

$-\lg b_H(\sigma_H) + \lg b'_H(\sigma_H)$

$=$ $\langle$ Lemma 2.2 (below), properties of $\lg$ $\rangle$

$-\lg \Pr_{\delta_A}(o) + \lg \Pr_{\delta_S}(o)$

$=$ $\langle$ Definition of $\mathcal{I}$ $\rangle$

$\mathcal{I}_{\delta_A}(o) - \mathcal{I}_{\delta_S}(o)$

$\square$

Lemma 2.2. $b'_H(\sigma_H) = b_H(\sigma_H) \cdot \frac{\delta_S(o)}{\delta_A(o)}$.

Proof.

$b'_H(\sigma_H)$

$=$ $\langle$ Definition of b'_H in experiment protocol $\rangle$

$((\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H)|o) \upharpoonright H)(\sigma_H)$

$=$ $\langle$ Definition of $\delta \upharpoonright H$ $\rangle$

$(\sum \sigma \,:\, \sigma \upharpoonright H = \sigma_H \,:\, (\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H)|o)(\sigma))$

$=$ $\langle$ Definition of $\delta|o$ $\rangle$

$(\sum \sigma \,:\, \sigma \upharpoonright H = \sigma_H \;\wedge\; \sigma \upharpoonright L = o \,:\, \frac{\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H)(\sigma)}{(\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H) \upharpoonright L)(o)})$

$=$ $\langle$ One-point rule: $\sigma = o \cup \sigma_H$ $\rangle$

$\frac{\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H)(o \cup \sigma_H)}{(\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H) \upharpoonright L)(o)}$

$=$ $\langle$ Definition of δ_A $\rangle$

$\frac{1}{\delta_A(o)} \cdot \llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H)(o \cup \sigma_H)$

$=$ $\langle$ Definition of $\llbracket S \rrbracket \delta$ $\rangle$

$\frac{1}{\delta_A(o)} \cdot (\sum \sigma' : (\dot{\sigma}_L \otimes b_H)(\sigma') \cdot ([\![S]\!]\sigma')(o \cup \sigma_H))$

$=$ $\langle$ Definition of $\otimes$, point mass $\rangle$

$\frac{1}{\delta_A(o)} \cdot (\sum \sigma' : \sigma' \restriction L = \sigma_L : b_H(\sigma' \restriction H) \cdot ([\![S]\!](\dot{\sigma}_L \otimes (\dot{\sigma}' \restriction H)))(o \cup \sigma_H))$

$=$ $\langle$ High input is immutable $\rangle$

$\frac{1}{\delta_A(o)} \cdot (\sum \sigma' : \sigma' \restriction L = \sigma_L \ \wedge \ \sigma' \restriction H = \sigma_H : b_H(\sigma' \restriction H)$

$\cdot ([\![S]\!](\dot{\sigma}_L \otimes (\dot{\sigma}' \restriction H)))(o \cup \sigma_H))$

$=$ $\langle$ One-point rule: $\sigma' = \sigma_L \cup \sigma_H$ $\rangle$

$\frac{1}{\delta_A(o)} \cdot b_H(\sigma_H) \cdot ([\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}'_H))(o \cup \sigma_H)$

$=$ $\langle$ High input is immutable, Definition of $\delta \restriction L$ $\rangle$

$\frac{1}{\delta_A(o)} \cdot b_H(\sigma_H) \cdot (([\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}'_H)) \restriction L)(o)$

$=$ $\langle$ Definition of δ_S $\rangle$

$b_H(\sigma_H) \cdot \dfrac{\delta_S(o)}{\delta_A(o)}$

□

Theorem 2.3. Let $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle$. Then:

$$\mathcal{Q}(\mathcal{E}, b'_H) = k \ \equiv \ b'_H(\sigma_H) = 2^k \cdot b_H(\sigma_H).$$

Proof.

$\mathcal{Q}(\mathcal{E}, b'_H) = k$

$\equiv$ $\langle$ Definition of $\mathcal{Q}$ $\rangle$

$D(b_H \rightharpoonup \dot{\sigma}_H) - D(b'_H \rightharpoonup \dot{\sigma}_H) = k$

$\equiv$ $\langle$ Definition of D $\rangle$

$-(\lg b_h(\sigma_H) - \lg b'_H(\sigma_H)) = k$

$\equiv$ $\langle$ Arithmetic, properties of $\log$ $\rangle$

$b'_H(\sigma_H) = 2^k \cdot b_H(\sigma_H)$

□

Theorem 2.4. $S \in \mathbf{Det} \implies \forall \mathcal{E}, b'_H \in \mathcal{B}(\mathcal{E}) \mathrel{.} \mathcal{Q}(\mathcal{E}, b'_H) \geq 0.$

Proof. Assume $S \in \mathbf{Det}$ and let $\mathcal{E}$, b'_H be arbitrary.

$\mathcal{Q}(\mathcal{E}, b'_H) \geq 0$

$\equiv$ $\langle$ Definition of $\mathcal{Q}$, arithmetic $\rangle$

$D(b_H \rightarrowtriangle \dot{\sigma}_H) \geq D(b'_H \rightarrowtriangle \dot{\sigma}_H)$

$\equiv$ $\langle$ Definition of D, arithmetic $\rangle$

$\lg b(\sigma_H) \leq \lg b'(\sigma_H)$

$\equiv$ $\langle$ Lemma 2.3 (below), $\lg$ is monotonic on $(0,1]$, admissibility of b $\rangle$

$true$

□

Lemma 2.3. *Assume* $S \in \mathbf{Det}$ *and let* $\mathcal{E}$, b'_H *be arbitrary. Then:*

$$b(\sigma_H) \leq b'(\sigma_H).$$

Proof.

$b'(\sigma_H)$

$=$ $\langle$ Definition of b' $\rangle$

$([\![S]\!](\dot{\sigma}_L \otimes b_H)|o \upharpoonright H)(\sigma_H)$

$=$ $\langle$ Definition of $\upharpoonright H$, application to σ_H, one-point rule $\rangle$

$(\sum \sigma'_L : ([\![S]\!](\dot{\sigma}_L \otimes b_H)|o)(\sigma'_L \cup \sigma_H))$

$=$ $\langle$ Definition of $|$, one-point rule $\rangle$

$$\frac{[\![S]\!](\dot{\sigma}_L \otimes b_H)(o \cup \sigma_H)}{([\![S]\!](\dot{\sigma}_L \otimes b_H) \upharpoonright L)(o)}$$

$=$ $\langle$ High input is immutable $\rangle$

$$\frac{b(\sigma_H) \cdot [\![S]\!](\dot{\sigma}_L \otimes b_H)(o \cup \sigma_H)}{([\![S]\!](\dot{\sigma}_L \otimes b_H) \upharpoonright L)(o)}$$

$=$ $\langle$ Output of S is a point mass (see below), let x be the denominator $\rangle$

$$\frac{b(\sigma_H) \cdot 1}{x}$$

$\geq$ $\langle$ Admissibility of b implies $x \in (0, 1]$, arithmetic $\rangle$

$b(\sigma_H)$

To see that the output of S is a point mass, let o be the observation producing b'. It is straightforward to check that if $S \in$ **Det**, then $[\![S]\!]\sigma$ is the point mass at σ', where σ' is the state produced by the standard denotational semantics of **while** programs, such as Winskel's [125]. So the output of $[\![S]\!](\sigma_L \cup \sigma_H)$ is the point mass at $o \cup \sigma_H$. □

Theorem 2.5. $\mathcal{Q}(\mathcal{E}, b') = \mathcal{L}(H_{in}, L_{in}, L_{out})$.

Proof.

$\mathcal{Q}(\mathcal{E}, b')$

$=$ $\langle$ Definition of $\mathcal{Q}$ $\rangle$

$D(b \rightharpoonup \dot{\sigma}_H) - D(b' \rightharpoonup \dot{\sigma}_H)$

$=\quad \langle$ Definition of D $\rangle$

$\mathcal{H}(b \upharpoonright (L \cup L^0 \cup H^0)) - \mathcal{H}(b \upharpoonright (L \cup L^0))$

$- (\mathcal{H}(b' \upharpoonright (L \cup L^0 \cup H^0)) - \mathcal{H}(b' \upharpoonright (L \cup L^0)))$

$=\quad \langle$ Definition of domain of b $\rangle$

$\mathcal{H}(b \upharpoonright (L^0 \cup H^0)) - \mathcal{H}(b \upharpoonright L^0) - (\mathcal{H}(b' \upharpoonright (L \cup L^0 \cup H^0)) - \mathcal{H}(b' \upharpoonright (L \cup L^0)))$

$=\quad \langle$ Definitions of H_{in}, L_{in}, L_{out}; b' is an output distribution $\rangle$

$\mathcal{H}(H_{in}, L_{in}) - \mathcal{H}(L_{in}) - (\mathcal{H}(H_{in}, L_{in}, L_{out}) - \mathcal{H}(L_{in}, L_{out}))$

$=\quad \langle$ Definition of conditional entropy $\rangle$

$\mathcal{H}(H_{in}|L_{in}) - \mathcal{H}(H_{in}|L_{in}, L_{out})$

$=\quad \langle$ Definition of $\mathcal{L}$ $\rangle$

$\mathcal{L}(H_{in}, L_{in}, L_{out})$

$\square$

Theorem 2.6 Let:

$$\begin{aligned}
\mathcal{E} &= \langle S, b_H, \sigma_H, \sigma_L \rangle, \\
\delta' &= [\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H), \\
e_H &= (([\![S]\!](\dot{\sigma}_L \otimes b_H))|(\delta' \upharpoonright L)) \upharpoonright H.
\end{aligned}$$

Then:

$$\mathcal{Q}_{\mathsf{E}}(\mathcal{E}) \quad \leq \quad \mathcal{Q}(\mathcal{E}, e_H).$$

Proof.

$\mathcal{Q}_{\mathsf{E}}(\mathcal{E})$

$=\quad \langle$ Definition of $\mathcal{Q}_{\mathsf{E}}$ $\rangle$

$\mathsf{E}_{o \in \delta' \restriction L}[\mathcal{Q}(\mathcal{E}, \mathcal{B}(\mathcal{E}, o))]$

$=$ ⟨ Definition of $\mathcal{Q}$, let $b'_H = \mathcal{B}(\mathcal{E}, o)$ ⟩

$\mathsf{E}_{o \in \delta' \restriction L}[D(b_H \twoheadrightarrow \dot{\sigma}_H) - D(b'_H \twoheadrightarrow \dot{\sigma}_H)]$

$=$ ⟨ Linearity of E ⟩

$D(b_H \twoheadrightarrow \dot{\sigma}_H) - \mathsf{E}_{o \in \delta' \restriction L}[D(b'_H \twoheadrightarrow \dot{\sigma}_H)]$

$\leq$ ⟨ Jensen's inequality and convexity of D [32] ⟩

$D(b_H \twoheadrightarrow \dot{\sigma}_H) - D(\mathsf{E}_{o \in \delta' \restriction L}[b'_H] \twoheadrightarrow \dot{\sigma}_H)$

$=$ ⟨ Lemma 2.4 ⟩

$D(b_H \twoheadrightarrow \dot{\sigma}_H) - D(e_H \twoheadrightarrow \dot{\sigma}_H)$

$=$ ⟨ Definition of $\mathcal{Q}$ ⟩

$\mathcal{Q}(\mathcal{E}, e_H)$

□

Lemma 2.4. *Let $\mathcal{E}$, δ', e_H be defined as in theorem 2.6. Let $b'_H = \mathcal{B}(\mathcal{E}, o)$, where $o \in \delta' \restriction L$. Then:*

$$\mathsf{E}_{o \in \delta' \restriction L}[b'_H] \quad = \quad e_H.$$

Proof. (by extensionality)

$\mathsf{E}_{o \in \delta' \restriction L}[b'_H](\sigma_H)$

$=$ ⟨ Definitions of E, b'_H ⟩

$((\sum o : (\delta' \restriction L)(o) \cdot \mathcal{B}(\mathcal{E}, o)))(\sigma_H)$

$=$ ⟨ Definition of $\mathcal{B}(\mathcal{E}, o)$ ⟩

$((\sum o : (\delta' \restriction L)(o) \cdot (((\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H))|o) \restriction H)))(\sigma_H)$

$=\quad \langle$ Definition of $\delta \upharpoonright H$, applying distribution to σ_H $\rangle$

$(\sum o : (\delta' \upharpoonright L)(o) \cdot ((\sum \sigma' : \sigma' \upharpoonright H = \sigma_H : ((\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H))|o)(\sigma'))))$

$=\quad \langle$ Definition of $\delta|o$, applying distribution to σ' $\rangle$

$(\sum o : (\delta' \upharpoonright L)(o) \cdot (\sum \sigma' : \sigma' \upharpoonright H = \sigma_H \wedge \sigma' \upharpoonright L = o : \frac{(\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H))(\sigma')}{(\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H) \upharpoonright L)(o)}))$

$=\quad \langle$ One-point rule $\rangle$

$(\sum o : (\delta' \upharpoonright L)(o) \cdot \frac{(\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H))(o \cup \sigma_H)}{(\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H) \upharpoonright L)(o)})$

$=\quad \langle$ Definition of $\delta \upharpoonright L$, applied to o $\rangle$

$(\sum o : (\delta' \upharpoonright L)(o) \cdot \frac{(\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H))(o \cup \sigma_H)}{(\sum \sigma' : \sigma' \upharpoonright L = o : \llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H)(\sigma'))})$

$=\quad \langle$ Let $\sigma = o \cup \sigma_H$, change of dummy: $o := \sigma$, definition of $=_L$ $\rangle$

$(\sum \sigma : \sigma \upharpoonright H = \sigma_H : (\delta' \upharpoonright L)(o) \cdot \frac{(\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H))(\sigma)}{(\sum \sigma' : \sigma' =_L \sigma : \llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H)(\sigma'))})$

$=\quad \langle$ Definition of $\delta|\delta_L$, applied to σ $\rangle$

$(\sum \sigma : \sigma \upharpoonright H = \sigma_H : (\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H)|(\delta' \upharpoonright L))(\sigma))$

$=\quad \langle$ Definition of $\delta \upharpoonright H$, applied to σ_H $\rangle$

$((\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H)|(\delta' \upharpoonright L)) \upharpoonright H)(\sigma_H)$

$=\quad \langle$ Definition of e_H $\rangle$

$e_H(\sigma_H)$

□

Theorem 2.7. Let $\mathcal{E} = \langle S, b_H, \delta_H, \sigma_L \rangle$, where $b_H = \delta_H$. Then:

$$\mathcal{Q}_{\mathsf{E}}(\mathcal{E}) \quad = \quad \mathcal{I}(H_{in}, L_{out}|L_{in}).$$

Proof. Consider the amount of flow resulting from a given high input σ_H, observation o, and postbelief b'_H. We calculate:

$$\mathcal{Q}(\langle S, b_H, \sigma_H, \sigma_L\rangle, b'_H)$$

$$= \quad \langle \text{ Definition of } \mathcal{Q} \text{ } \rangle$$

$$D(b_H \rightarrow \dot{\sigma}_H) - D(b'_H \rightarrow \dot{\sigma}_H)$$

$$= \quad \langle \text{ Definition of } D \text{ } \rangle$$

$$-\lg(b_H(\sigma_H)) + \lg(b'_H(\sigma_H))$$

$$= \quad \langle \text{ Log identity } \rangle$$

$$\lg \frac{b'_H(\sigma_H)}{b_H(\sigma_H)}$$

$$= \quad \langle \text{ Definition of } b'_H \text{ and } \delta_A \text{ } \rangle$$

$$\lg \frac{((\delta_A|o) \upharpoonright H)(\sigma_H)}{b_H(\sigma_H)}$$

$$= \quad \langle \text{ Lemma 2.5 } \rangle$$

$$\lg \frac{((\delta_A|o) \upharpoonright H)(\sigma_H)}{\delta_A(\sigma_H)}$$

It is now convenient to introduce notation for probability. Let $\Pr_\delta(E)$ denote the probability of event E according to distribution δ, and let $\Pr_\delta(E|F)$ denote $\Pr_{\delta|F}(E)$. Let h denote the event that the high input sampled from H_{in} is σ_h, let l denote the event that the low input sampled from L_{in} (which is actually a point mass) is σ_L, and let o denote the event that the observation sampled from L_{out} is o. Then $((\delta_A|o) \upharpoonright H)(\sigma_H)$ can be rewritten as $\Pr_{\delta_A}(h|o)$; and $\delta_A(\sigma_H)$, as $\Pr_{\delta_A}(h)$. We continue calculating:

$$\lg \frac{((\delta_A|o) \upharpoonright H)(\sigma_H)}{\delta_A(\sigma_H)}$$

$$= \quad \langle \text{ Rewriting using probability notation } \rangle$$

$$\lg \frac{\Pr_{\delta_A}(h|o)}{\Pr_{\delta_A}(h)}$$

$=$ $\langle\ \delta_A = \delta_A|\sigma_L\ \rangle$

$$\lg \frac{\Pr_{\delta_A|l}(h|o)}{\Pr_{\delta_A|l}(h)}$$

$=$ $\langle$ Rewriting using conditional probability notation $\rangle$

$$\lg \frac{\Pr_{\delta_A}(h|o,l)}{\Pr_{\delta_A}(h|l)}$$

$=$ $\langle$ Definition of conditional probability $\rangle$

$$\lg \frac{\Pr_{\delta_A}(h,o|l)}{\Pr_{\delta_A}(h|l)\cdot\Pr_{\delta_A}(o|l)}$$

Now take the expectation of the amount of flow with respect to observation o, which is distributed according to $\delta' = [\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H)$.

$$\mathsf{E}_o[\lg \frac{\Pr_{\delta_A}(h,o|l)}{\Pr_{\delta_A}(h|l)\cdot\Pr_{\delta_A}(o|l)}]$$

$=$ $\langle$ Definition of E $\rangle$

$$\textstyle\sum_o \Pr_{\delta'}(o)\cdot\lg \frac{\Pr_{\delta_A}(h,o|l)}{\Pr_{\delta_A}(h|l)\cdot\Pr_{\delta_A}(o|l)}$$

$=$ $\langle\ \delta' = \delta_A|h,l$; conditional probability notation $\rangle$

$$\textstyle\sum_o \Pr_{\delta_A}(o|h,l)\cdot\lg \frac{\Pr_{\delta_A}(h,o|l)}{\Pr_{\delta_A}(h|l)\cdot\Pr_{\delta_A}(o|l)}$$

Again take the expectation, now with respect to high input h, whose distribution is δ_H:

$$\mathsf{E}_h[\textstyle\sum_o \Pr_{\delta_A}(o|h,l)\cdot\lg \frac{\Pr_{\delta_A}(h,o|l)}{\Pr_{\delta_A}(h|l)\cdot\Pr_{\delta_A}(o|l)}]$$

$=$ $\langle$ Definition of E $\rangle$

$$\textstyle\sum_h \Pr_{\delta_H}(h)\cdot\sum_o \Pr_{\delta_A}(o|h,l)\cdot\lg \frac{\Pr_{\delta_A}(h,o|l)}{\Pr_{\delta_A}(h|l)\cdot\Pr_{\delta_A}(o|l)}$$

$=$ $\langle\ \delta_H = b_H\ \rangle$

$$\sum_h \Pr_{b_H}(h) \cdot \sum_o \Pr_{\delta_A}(o|h,l) \cdot \lg \frac{\Pr_{\delta_A}(h,o|l)}{\Pr_{\delta_A}(h|l) \cdot \Pr_{\delta_A}(o|l)}$$

$=$ $\langle$ Lemma 2.5 $\rangle$

$$\sum_h \Pr_{\delta_A}(h) \cdot \sum_o \Pr_{\delta_A}(o|h,l) \cdot \lg \frac{\Pr_{\delta_A}(h,o|l)}{\Pr_{\delta_A}(h|l) \cdot \Pr_{\delta_A}(o|l)}$$

$=$ $\langle$ $\delta_A|l = \delta_A$; conditional probability notation $\rangle$

$$\sum_h \Pr_{\delta_A}(h|l) \cdot \sum_o \Pr_{\delta_A}(o|h,l) \cdot \lg \frac{\Pr_{\delta_A}(h,o|l)}{\Pr_{\delta_A}(h|l) \cdot \Pr_{\delta_A}(o|l)}$$

$=$ $\langle$ Distributivity $\rangle$

$$\sum_h \sum_o \Pr_{\delta_A}(h|l) \cdot \Pr_{\delta_A}(o|h,l) \cdot \lg \frac{\Pr_{\delta_A}(h,o|l)}{\Pr_{\delta_A}(h|l) \cdot \Pr_{\delta_A}(o|l)}$$

$=$ $\langle$ Definition of conditional probability $\rangle$

$$\sum_h \sum_o \Pr_{\delta_A}(h,o|l) \cdot \lg \frac{\Pr_{\delta_A}(h,o|l)}{\Pr_{\delta_A}(h|l) \cdot \Pr_{\delta_A}(o|l)}$$

$=$ $\langle$ Definition of conditional probability $\rangle$

$$\sum_h \sum_o \frac{\Pr_{\delta_A}(h,o,l)}{\Pr_{\delta_A}(l)} \cdot \lg \frac{\Pr_{\delta_A}(h,o|l)}{\Pr_{\delta_A}(h|l) \cdot \Pr_{\delta_A}(o|l)}$$

$=$ $\langle$ δ_A is a point mass at l, twice $\rangle$

$$\sum_l \sum_h \sum_o \Pr_{\delta_A}(h,o,l) \cdot \lg \frac{\Pr_{\delta_A}(h,o|l)}{\Pr_{\delta_A}(h|l) \cdot \Pr_{\delta_A}(o|l)}$$

$=$ $\langle$ Definition of mutual information $\rangle$

$$\mathcal{I}(H_{in}, L_{out}|L_{in})$$

□

Lemma 2.5. $b_H = \delta_A \restriction H$.

Proof. Let σ_H be arbitrary, and let $b = \dot{\sigma}_L \otimes b_H$ be the attacker's belief about the entire (low and high) state. We calculate:

$$
\begin{array}{ll}
& (\delta_A \restriction H)(\sigma_H) \\
= & \quad \langle \text{ Definition of } \delta_A \rangle \\
& ([\![S]\!](\dot{\sigma}_L \otimes b_H) \restriction H)(\sigma_H) \\
= & \quad \langle \text{ Definition of } b \rangle \\
& (([\![S]\!]b) \restriction H)(\sigma_H) \\
= & \quad \langle \text{ Definition of } \restriction H \rangle \\
& (\sum \sigma \,:\, \sigma \restriction H = \sigma_H \,:\, ([\![S]\!]b)(\sigma)) \\
= & \quad \langle \text{ Definition of } [\![S]\!]\delta \rangle \\
& (\sum \sigma \,:\, \sigma \restriction H = \sigma_H \,:\, (\sum \sigma' \,:\, b(\sigma') \cdot ([\![S]\!]\sigma')(\sigma))) \\
= & \quad \langle \text{ High input is immutable } \rangle \\
& (\sum \sigma \,:\, \sigma \restriction H = \sigma_H \,:\, (\sum \sigma' \,:\, \sigma' \restriction H = \sigma_H \,:\, b(\sigma') \cdot ([\![S]\!]\sigma')(\sigma))) \\
= & \quad \langle \text{ Commutativity, distributivity } \rangle \\
& (\sum \sigma' \,:\, \sigma' \restriction H = \sigma_H \,:\, b(\sigma') \cdot (\sum \sigma \,:\, \sigma \restriction H = \sigma_H \,:\, ([\![S]\!]\sigma')(\sigma))) \\
= & \quad \langle \text{ High input is immutable } \rangle \\
& (\sum \sigma' \,:\, \sigma' \restriction H = \sigma_H \,:\, b(\sigma') \cdot (\sum \sigma \,:\, ([\![S]\!]\sigma')(\sigma))) \\
= & \quad \langle\ S \text{ always terminates } \rangle \\
& (\sum \sigma' \,:\, \sigma' \restriction H = \sigma_H \,:\, b(\sigma') \cdot 1)
\end{array}
$$

$=$ $\langle$ Definition of $\restriction H$ $\rangle$

$(b \restriction H)(\sigma_H)$

$=$ $\langle$ Definition of b $\rangle$

$b_H(\sigma_H)$

Therefore $b_H = \delta_A \restriction H$ by extensionality.

$\square$

Theorem 2.9. $S \in \textbf{ObsDet} \equiv \forall \mathcal{E}, b'_H \in \mathcal{B}(\mathcal{E}) \,.\, \mathcal{Q}(\mathcal{E}, b'_H) = 0.$

Proof. By mutual implication.

($\Rightarrow$) Assume $S \in \textbf{ObsDet}$. Let $\mathcal{E} = \langle S, \sigma_L, \sigma_H, b_H, I\rangle$ and $b'_H \in \mathcal{B}(\mathcal{E})$ be arbitrary.

$\mathcal{Q}(\mathcal{E}, b'_H) = 0$

$\equiv$ $\langle$ Definition of $\mathcal{Q}$, arithmetic $\rangle$

$D(b_H \twoheadrightarrow \sigma_H) = D(b'_H \twoheadrightarrow \sigma_H)$

$\equiv$ $\langle$ Definition of D, arithmetic $\rangle$

$b_H(\sigma_H) = b'_H(\sigma_H)$

$\equiv$ $\langle$ Lemma 2.7 $\rangle$

$true$

This concludes the forward direction ($\Rightarrow$) of the proof.

($\Leftarrow$) By contrapositive. Assume $S \notin \mathbf{ObsDet}$. We need to show:

$$\exists \mathcal{E} = \langle S, \sigma_L, \sigma_H, b_H, I \rangle, b'_H \in \mathcal{B}(\mathcal{E}) \mathrel{.} \mathcal{Q}(\mathcal{E}, b'_H \neq 0$$

We calculate:

$$\begin{array}{cl}
 & S \notin \mathbf{ObsDet} \\
\equiv & \langle \text{ Definition of } \mathbf{ObsDet} \rangle \\
 & \neg \forall I, \sigma_L \exists \delta_L \forall \sigma_H \mathrel{.} [\![S]\!]_I(\dot{\sigma}_L \otimes \dot{\sigma}_H) \restriction L = \delta_L \\
\equiv & \langle \text{ Predicate calculus, change of dummy } \rangle \\
 & \exists \tilde{I}, \tilde{\sigma}_L \forall \tilde{\delta}_L \exists \tilde{\sigma}_H \mathrel{.} [\![S]\!]_{\tilde{I}}(\dot{\tilde{\sigma}}_L \otimes \dot{\tilde{\sigma}}_H) \restriction L \neq \tilde{\delta}_L \qquad (*)
\end{array}$$

Make the following definitions:

$$\begin{array}{rcl}
I & = & \tilde{I} \\
\sigma_L & = & \tilde{\sigma}_L \\
\sigma'_H & = & \text{arbitrary} \\
\delta' & = & [\![S]\!]_I(\dot{\sigma}_L \otimes \dot{\sigma}'_H) \\
\delta'_L & = & \delta' \restriction L \\
\sigma_H & = & \text{the } \tilde{\sigma}_H \text{ guaranteed by formula (*) above when } \tilde{\delta}_L = \delta'_L \\
\delta & = & [\![S]\!]_I(\dot{\sigma}_L \otimes \dot{\sigma}_H) \\
\delta_L & = & \delta \restriction L
\end{array}$$

And let b_H be the belief mapping σ_H to $1/2$ and σ'_H to $1/2$.

We have now defined all the variables in experiment $\mathcal{E}$, but we need to define $b'_H \in \mathcal{B}(\mathcal{E})$. To that end, we calculate attacker prediction δ_A:

δ_A

$= \quad \langle$ Definition of prediction $\rangle$

$[\![S]\!]_I(\dot{\sigma}_L \otimes b_H)$

$= \quad \langle$ Definition of $[\![S]\!]\delta$ $\rangle$

$1/2 \cdot [\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H) + 1/2 \cdot [\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}'_H)$

$= \quad \langle$ Definition of δ,δ' $\rangle$

$1/2 \cdot (\delta + \delta')$

To define b'_H, we also need an observation o. Note that, by formula (2.6), $\delta_L \neq \delta'_L$, so there is some low state σ'_L such that $\delta_L(\sigma'_L) \neq \delta'_L(\sigma'_L)$. Assume, without loss of generality, that $\delta_L(\sigma'_L) > \delta'_L(\sigma'_L)$. Let o be σ'_L. But in order for o to be an observation, it must be that $o \in [\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H)$, which implies that $[\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H)(o) > 0$. This is guaranteed by the fact that $\delta_L(o) > \delta'_L(o)$, and that $\delta'_L(o) \geq 0$.

We can now calculate b'_H:

b'_H

$= \quad \langle$ Definition of b'_H experiment protocol $\rangle$

$\delta_A | o \upharpoonright H$

$= \quad \langle$ Definition of δ_A $\rangle$

$1/2 \cdot (\delta + \delta') | o \upharpoonright H$

With all these definitions, we can prove the desired result:

$\mathcal{Q}(\mathcal{E}, b'_H) \neq 0$

$\equiv$ $\langle$ Definition of $\mathcal{Q}$, arithmetic $\rangle$

$D(b_H \multimap \sigma_H) \neq D(b'_H \multimap \sigma_H)$

$\equiv$ $\langle$ Definition of D, arithmetic $\rangle$

$b_H(\sigma_H) \neq b'_H(\sigma_H)$

$\equiv$ $\langle$ Lemma 2.9 $\rangle$

$true$

$\square$

Lemma 2.6.

$$S \in \mathbf{ObsDet} \quad \Longrightarrow \quad \forall I \,.\, \forall \sigma_L \,.\, \exists \delta_L \,.\, \forall \delta_H \,.\, \|\delta_H\| = 1 \;\Longrightarrow\; [\![S]\!]_I(\dot{\sigma}_L \otimes \dot{\sigma}_H) \upharpoonright L = \delta_L.$$

Proof. Assume $S \in$ **ObsDet**. Let I, σ_L be arbitrary. Let δ_L be the distribution guaranteed to exist by the definition of **ObsDet**. Let δ_H be arbitrary such that $\|\delta_H\| = 1$.

$[\![S]\!]_I(\dot{\sigma}_L \otimes \dot{\sigma}_H) \upharpoonright L$

$=$ $\langle$ Definition of $[\![S]\!]\delta$ $\rangle$

$((\sum \sigma_H \,:\, \delta_H(\sigma_H) \cdot [\![S]\!]_I(\sigma_L \cup \sigma_H))) \upharpoonright L$

$=$ $\langle \upharpoonright L$ distributes over $+, \cdot$ $\rangle$

$(\sum \sigma_H \,:\, \delta_H(\sigma_H) \cdot [\![S]\!]_I(\sigma_L \cup \sigma_H) \upharpoonright L)$

$=$ $\langle$ $S \in$ **ObsDet**, definition of δ_L $\rangle$

$(\sum \sigma_H \,:\, \delta_H(\sigma_H) \cdot \delta_L)$

$=$ $\langle$ Distributivity, definition of $\|\delta\|$ $\rangle$

$\delta_L \cdot \|\delta_H\|$

$=$ $\langle$ Assumed $\|\delta_H\| = 1$ $\rangle$

δ_L

□

Lemma 2.7. *Assume $S \in$ **ObsDet**. Let $\mathcal{E} = \langle S, \sigma_L, \sigma_H, b_H, I\rangle$ and $b'_H \in \mathcal{B}(\mathcal{E})$ be arbitrary. Then:*

$$b_H \quad = \quad b'_H.$$

Proof. Let $\delta_A = [\![S]\!]_I(\dot{\sigma}_L \otimes b_H)$. Let $o \in [\![S]\!]_I(\dot{\sigma}_L \otimes \dot{\sigma}_H) \upharpoonright L$.

b'_H

$=$ $\langle$ Definition of b'_H in experiment protocol $\rangle$

$(\delta_A | o) \upharpoonright H$

$=$ $\langle$ Definition of $\upharpoonright H$ $\rangle$

$\lambda\sigma_H \,.\, (\sum \sigma' \,:\, \sigma' \upharpoonright H = \sigma_H \,:\, (\delta_A|o)(\sigma'))$

$=$ $\langle$ Definition of $\delta|o$ $\rangle$

$\lambda\sigma_H \,.\, (\sum \sigma' \,:\, \sigma' \upharpoonright H = \sigma_H \,:\, \text{if } (\sigma' \upharpoonright L) = o \text{ then } \frac{\delta_A(\sigma')}{(\delta_A|L)(o)} \text{ else } 0)$

$=$ $\langle$ Lemma 2.6 $\rangle$

$\lambda\sigma_H \,.\, (\sum \sigma' \,:\, \sigma' \upharpoonright H = \sigma_H \,:\, \text{if } (\sigma' \upharpoonright L) = o \text{ then } \frac{\delta_A(\sigma')}{\delta_L(o)} \text{ else } 0)$

$=$ $\langle$ One-point rule $\rangle$

$\lambda\sigma_H \,.\, \frac{\delta_A(o\cup\sigma_H)}{\delta_L(o)}$

$=$ $\langle$ Lemma 2.8 $\rangle$

$\lambda\sigma_H \,.\, \frac{b_H(\sigma_H)\cdot\delta_L(o)}{\delta_L(o)}$

$=$ $\langle$ Arithmetic, η-reduction $\rangle$

b_H

$\square$

Lemma 2.8. *Assume the definitions in lemma 2.7 and its proof. Then:*

$$\delta_A(o\cup\sigma_H) \quad = \quad b_H(\sigma_H)\cdot\delta_L(o).$$

Proof.

$\delta_A(o\cup\sigma_H)$

$=$ $\langle$ Definition of δ_A $\rangle$

$[\![S]\!]_I(\dot{\sigma}_L\otimes b_H)(o\cup\sigma_H)$

$=$ $\langle$ Definition of $[\![S]\!]\delta$ $\rangle$

$(\sum\, \sigma' \,:\, (\dot{\sigma}_L\otimes b_H)(\sigma')\cdot([\![S]\!]_I\sigma')(o\cup\sigma_H))$

$=$ $\langle$ Definition of $\otimes$, one-point rule $\rangle$

$(\sum\, \sigma'_H \,:\, b_H(\sigma'_H)\cdot([\![S]\!]_I(\sigma_L\cup\sigma'_H))(o\cup\sigma_H))$

$=$ $\langle$ Immutable high input, one-point rule $\rangle$

$b_H(\sigma_H)\cdot([\![S]\!]_I(\sigma_L\cup\sigma_H))(o\cup\sigma_H)$

$=$ $\langle$ Immutable high input, definition of $\upharpoonright L$ $\rangle$

$b_H(\sigma_H)\cdot(([\![S]\!]_I(\sigma_L\cup\sigma_H))\upharpoonright L)(o)$

$=\quad \langle\ S \in \textbf{ObsDet}$, definition of $\delta_L\ \rangle$

$b_H(\sigma_H) \cdot \delta_L(o)$

$\square$

Lemma 2.9. *Assume the definitions in the contrapositive proof of theorem 2.9. Then:*

$$b_H(\sigma_H) \quad \neq \quad b'_H(\sigma_H).$$

Proof. First we calculate $b'_H(\sigma_H)$:

$b'_H(\sigma_H)$

$=\quad \langle$ Definition of $b'_H\ \rangle$

$(\delta_A|o\restriction H)(\sigma_H)$

$=\quad \langle$ Calculation of δ_A in theorem 2.9 $\rangle$

$(1/2 \cdot (\delta + \delta')|o\restriction H)(\sigma_H)$

$=\quad \langle$ Definition of $\delta\restriction H$, one-point rule, D defined below $\rangle$

$(\sum \sigma_L\ :\ (1/2 \cdot (\delta + \delta')|o)(\sigma_L \cup \sigma_H)/D)$

$=\quad \langle$ Definition of $\delta|o$, one-point rule $\rangle$

$1/2 \cdot (\delta + \delta')(o \cup \sigma_H)/D$

$=\quad \langle$ Definition of $+$ for distributions $\rangle$

$1/2 \cdot (\delta(o \cup \sigma_H) + \delta'(o \cup \sigma'_H))/D$

$=\quad \langle$ Definition of δ', immutability of H input $\rangle$

$1/2 \cdot \delta(o \cup \sigma_H)/D$

Quantity D is defined to be $1/2 \cdot (\delta(o \cup \sigma_H) + \delta'(o \cup \sigma'_H))$. Similarly, we can calculate $b'_H(\sigma'_H) = 1/2 \cdot \delta(o \cup \sigma'_H)/D$.

We next calculate $\delta_L(o)$:

$$
\begin{array}{ll}
 & \delta_L(o) \\
= & \quad \langle \text{ Definition of } \delta_L \text{ and projection } \rangle \\
 & (\sum \sigma_H \,:\, \delta(o \cup \sigma_H)) \\
= & \quad \langle \text{ Definition of } \delta\text{, immutability of high input, one-point rule } \rangle \\
 & \delta(o \cup \sigma_H)
\end{array}
$$

Similarly, $\delta'_L(o) = \delta'(o \cup \sigma'_H)$. By the definition of o we have $\delta_L(o) \neq \delta'_L(o)$, so $\delta(o \cup \sigma_H) \neq \delta'(o \cup \sigma'_H)$. Thus:

$$
\begin{array}{ll}
 & b'_H(\sigma'_H) \\
= & \quad \langle \text{ Calculated value of } b'_H(\sigma'_H) \text{ } \rangle \\
 & 1/2 \cdot \delta(o \cup \sigma'_H)/D \\
\neq & \quad \langle \text{ Above inequality } \rangle \\
 & 1/2 \cdot \delta(o \cup \sigma_H)/D \\
= & \quad \langle \text{ Calculated value of } b'_H(\sigma_H) \text{ } \rangle \\
 & b'_H(\sigma_H)
\end{array}
$$

Finally, note that by the immutability of high input, the only high states with non-zero mass in b'_H are σ_H and σ'_H. If $b'_H(\sigma_H) = 1/2$, we would be forced to conclude $b'_H(\sigma'_H) = 1/2$ because the mass in a belief must sum to 1. But this would contradict the previous calculation. So $b'_H(\sigma_H) \neq 1/2$. Thus, since $b_H(\sigma_H) = 1/2$, we conclude $b_H(\sigma_H) \neq b'_H(\sigma_H)$.

$\square$

Theorem 3.2 $D(b \rightarrow \dot{D}) = \mathcal{Q}(\langle A, b, D, q\rangle, b') + \mathcal{S}_P(\langle A, b, D, q\rangle, b')$

Proof. The quantity of leakage is

$$\mathcal{Q}(\langle A, b, D, q\rangle, b') = D(b \rightarrow \dot{D}) - D(b' \rightarrow \dot{D}).$$

And the amount of program suppression is

$$\begin{aligned} \mathcal{S}_P(\langle A, b, D, q\rangle, b') &= D((b'|q) \upharpoonright TI \rightarrow \dot{D}) \\ &= D(b' \rightarrow \dot{D}). \end{aligned}$$

The equality follows because q is already contained in the attacker's observation, so b' has already been conditioned on q; and because restricting b' to trusted inputs is here equivalent to restricting to secret inputs (i.e., the actual database contents), and this has already been done by the experiment protocol that produced b'.

Substituting and rewriting, we have

$$D(b \rightarrow \dot{D}) = \mathcal{Q}(\langle A, b, D, q\rangle, b') + \mathcal{S}_P(\langle A, b, D, q\rangle, b'). \qquad \square$$

CHAPTER 3
QUANTIFICATION OF INTEGRITY

Computer security policies often involve integrity requirements for information and other system resources—for example, that electronic data must correctly represent what appears in paper sources [37, glossary entry "data integrity"], that information may be modified only by authorized programs and authorized users [26], or that inputs to a program must be validated before being used to change system state external to the program, such as the filesystem [122, p. 356]. This last example can be interpreted as an information-flow security policy in which information from (attacker-controlled) inputs can be considered *untrusted*, whereas the system state should contain only *trusted* information: information flow from untrusted to trusted is prohibited unless it passes a validation procedure. *Taint analysis* [75, 93, 112, 122, 128] enforces a similar information-flow policy. Untrusted information is considered to be *tainted*; and trusted information, *untainted*. If information flows from tainted sources to a sink that is supposed to be untainted, *contamination* of the sink has occurred.

In some scenarios, a qualitative integrity policy might be overly restrictive. If the attacker can cause only a little contamination, a flow from tainted to untainted (i.e., untrusted to trusted) might be acceptable. Thus, quantitative integrity policies would be useful in characterizing security.

Since confidentiality and integrity are information-flow duals [15], previous models for quantification of information-flow confidentiality [11, 24, 35, 49, 76, 80, 88] seem likely to apply to quantification of integrity. In particular, the integrity policy "information is prohibited to flow from untrusted to trusted" is the dual of the confidentiality policy "information is prohibited to flow from secret to public," which is the kind of qualitative policy that previous work—and chapter 2—has made quantitative. Here, we adapt the results of chapter 2 to quantify contamination with accuracy of belief.

Besides contamination, there is another, distinct aspect of quantitative integrity. In the information-theoretic model of communication channels [32], a sender sends messages through a noisy channel to a receiver. The receiver cannot observe the sender's inputs or the noise but must attempt to determine what message was sent. A standard question to ask is: "how much information is transmitted over the channel?" When information is lost because of noise, information has been *suppressed*; noise thus damages the integrity of the information. Here, we show that suppression and transmission can be quantified by using accuracy of beliefs. We also examine error-correcting codes and show that, as we would expect, they reduce suppression of information. Moreover, analysis of suppression is applicable with programs in general, not just programs that model communication channels.

Contamination and suppression are not necessarily disjoint: A program that takes t as trusted input and u as untrusted input, then outputs pair (t, u) as trusted output, exhibits contamination—because output (t, u) is obviously affected by an untrusted input u—but does not exhibit suppression. A program that instead outputs $t \oplus n$, where $\oplus$ is exclusive-or and n is randomly generated noise, exhibits suppression but not contamination. And a program that outputs $t \oplus u$ exhibits both.

Quantifying confidentiality and integrity simultaneously is useful for understanding the security of a statistical database, which contains information about individuals and should respond to queries in a way that protects the privacy of those individuals. The queries and responses might involve statistics (e.g., sums or averages) computed from individuals' information. One mechanism that enforces this privacy policy is the addition of randomly generated noise to the underlying data or to the response [35]; the database is responding with information that has been deliberately suppressed to improve confidentiality. The quantitative frameworks we have developed for confidentiality and integrity can be used to analyze this enforcement mechanism.

This chapter proceeds as follows. Models and metrics for quantification of contamination and suppression are given in §3.1 and §3.2. These metrics are applied in §3.3 and §3.4 to error-correcting codes and statistical databases. The duality between confidentiality and integrity is explored in §3.5. Related work is discussed in §3.6, and §3.7 concludes. Most proofs are delayed from the main body to appendix 3.A.

3.1 Quantification of Contamination

Three agents are involved in execution of a program: a system, a user, and an attacker.[1] The *system* executes a program, whose variables are categorized as *input*, *output*, or *internal*. Input variables may only be read by the program, output variables may only be written by the program, and internal variables may be read and written but are never be observed by any agent except the system itself. The *user* and the *attacker* supply inputs by writing the initial values of input variables. These agents receive outputs by reading the final values of output variables. Since the attacker is *untrusted*, or low integrity, variables read and written by the attacker are labeled U. Likewise, the user is *trusted* and the user's variables are labeled T. The channels between agents and the program are depicted in figure 3.1.

[1]In chapter 2, we modeled only the system and the attacker. We further discuss the addition of the user in §3.1.1 and §3.5.

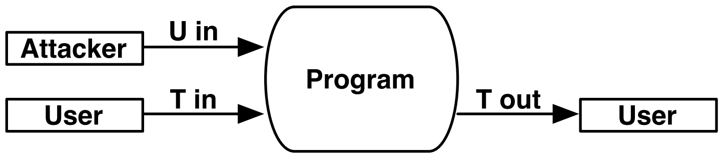

Figure 3.1: Channels in contamination experiment

Our goal is to quantify the amount of information about untrusted inputs that the user learns by observing trusted outputs. This goal entails two restrictions on the user's access to variables. First, the user should not be allowed to read untrusted inputs—otherwise, the user could learn all the untrusted information without observing any outputs. Second, the user should not be allowed to read untrusted outputs, because we are interested only in the information the user learns from trusted outputs. In addition to these restrictions, for simplicity, we do not allow the user to write untrusted inputs (although this would be possible to model). So the user may access only the trusted variables.

Similarly, the attacker may access only the untrusted variables. The attacker may not write trusted inputs because he is untrusted. And for simplicity, we do not allow the attacker to read trusted inputs or outputs. However, since flow from trusted to untrusted need not be prohibited, it would be possible to allow and to model such reads.

Note that these access rules agree with the Biba integrity model [15] in that they prohibit reading up (i.e., the user cannot read untrusted information) and writing down (i.e., the attacker cannot write trusted information).

3.1.1 Contamination Experiment Protocol

Users cannot directly observe untrusted inputs, thus users are uncertain about them. A user's *belief* characterizes this uncertainty. Note that it is now the user who holds beliefs—not the attacker, who held beliefs about secret inputs in the model of chapter 2. Recall (from §2.1) that beliefs are held about program *states*, which map variables to values. Previously, a state could be decomposed into two parts: its *high projection*, containing just the secret variables, and its *low projection*, containing just the public variables. Now, since we are concerned with integrity, we instead decompose a state into a *trusted projection*, containing just the trusted variables, and an *untrusted projection*, containing just the untrusted variables. The trusted projection of state σ is denoted $\sigma \upharpoonright T$; and the untrusted projection, $\sigma \upharpoonright U$. Probability distributions, hence beliefs, can likewise be projected. Previously, beliefs were probability distributions over the high projection of states; now, beliefs are probability distributions over the untrusted projection of states.

A *contamination experiment* describes how a user revises his beliefs about untrusted inputs. During an experiment, the user interacts with a system and observes trusted outputs. The protocol for contamination experiments is given in figure 3.2 and is explained below.[2]

Formally, a contamination experiment $\mathcal{E}$ is described by a tuple,

$$\mathcal{E} = \langle S, b_U, \sigma_U, \sigma_T \rangle,$$

where S is the program executed by the system, σ_U is the untrusted projection of the initial state, and σ_T is the trusted projection of the initial state. For simplicity,

[2]This protocol is essentially identical to the protocol for confidentiality experiments in figure 2.2. The changes are (i) the introduction of the user as an agent, (ii) the reversal of the roles of the user and attacker, and (iii) the substitution of "trusted" for "low" and "untrusted" for "high."

A contamination experiment $\mathcal{E} = \langle S, b_U, \sigma_U, \sigma_T \rangle$ is conducted as follows.

1. The user chooses a prebelief b_U about the untrusted state.
2. (a) The attacker picks an untrusted state σ_U.
   (b) The user picks a trusted state σ_T.
3. The user predicts the output distribution: $\delta'_P = [\![S]\!](\dot{\sigma}_T \otimes b_U)$.
4. The system executes program S, producing a state $\sigma' \in \delta'$ as output, where $\delta' = [\![S]\!](\dot{\sigma}_T \otimes \dot{\sigma}_U)$. The user observes the trusted projection of the output state: $o = \sigma' \upharpoonright T$.
5. The user infers a postbelief: $b'_U = (\delta'_P | o) \upharpoonright U$.

Figure 3.2: Contamination experiment protocol

assume that S always terminates.[3] Also assume that the attacker and user know the code of program S.

The user's *prebelief* b_U, characterizing his uncertainty about untrusted inputs at the beginning of the experiment, may be chosen arbitrarily.[4] The attacker chooses σ_U, the untrusted projection of the initial state, and the user chooses σ_T, the trusted projection of the initial state. Using the semantics of S along with prebelief b_U as a distribution on untrusted input, the user conducts a "thought experiment" to generate a *prediction* δ'_P of the output distribution:

$$\delta'_P = [\![S]\!](\dot{\sigma}_T \otimes b_U).$$

Program S is executed by the system. The distribution on output states produced by that execution is δ':

$$\delta' = [\![S]\!](\dot{\sigma}_T \otimes \dot{\sigma}_U).$$

[3]This assumption can be eliminated by using the technique described in §2.2.4. Also, recall that in confidentiality experiments we assumed that S did not modify any of the secret (high) projection of the state, because the initial secret values needed to be preserved in the final state. To remove this restriction, §2.2.4 described a technique for preserving a copy of the untrusted component of the state. But here, we have already introduced an alternate solution—the immutable inputs preserve such a copy. Thus copying is not needed here.

[4]As with confidentiality, an *admissibility restriction* (c.f. §2.1.4) can rule out nonsensical prebeliefs.

The user makes an *observation*, which is the trusted projection of an output state sampled from δ'. We write $\sigma' \in \delta'$ to denote that σ' is in the support of (i.e., has positive frequency according to) δ'. The observation o resulting from σ' is

$$o = \sigma' \upharpoonright T.$$

Finally, the user's postbelief b'_U is the untrusted projection of the distribution that results from conditioning prediction δ'_P on observation o:

$$b'_U = (\delta'_P | o) \upharpoonright U.$$

Postbelief b'_U characterizes the user's uncertainty about the untrusted inputs at the end of the experiment.

3.1.2 Contamination Metric

Define the amount of information flow $\mathcal{Q}_{con}$ caused by outcome b'_U of experiment $\mathcal{E}$ as the improvement in the accuracy of the user's belief:

$$\mathcal{Q}_{con}(\mathcal{E}, b'_U) \quad \triangleq \quad D(b_U \rightarrow \dot{\sigma}_U) - D(b'_U \rightarrow \dot{\sigma}_U).$$

Let D be instantiated with relative entropy as in chapter 2. Thus the unit of measurement for $\mathcal{Q}_{con}$ is (information-theoretic) bits.

As an example of quantification of contamination, consider the following program:

$$t2 := t1 + u$$

Variables $t1$ and $t2$ are trusted, whereas variable u is untrusted. Suppose that $t1$ and u are one-bit variables—that is, they can store either 0 or 1—but that $t2$ can store any integer. Let the user have a uniform prebelief b_U about the value of u. Based on his knowledge of $t1$, the user will correctly infer the value of u by observing $t2$. For example, if $\sigma_T(t1) = 0$ and $\sigma_U(u) = 1$, then observation o will be that $t2 = 1$, and postbelief b'_U will assign state $(u \mapsto 1)$ probability 1. Quantity of flow $\mathcal{Q}_{con}$ is thus 1. This amount is intuitively sensible: one bit of untrusted information, the value of u, has contaminated the trusted output.

More generally, we can show that $\mathcal{Q}_{con}$ correctly quantifies the information contained in an observation o about untrusted input σ_U. Let $\delta_Y = [\![S]\!](\dot{\sigma}_T \otimes \dot{\sigma}_U) \upharpoonright T$ be the system's distribution on trusted outputs, and let $\delta_U = [\![S]\!](\dot{\sigma}_T \otimes b_U) \upharpoonright T$ be the user's distribution on trusted outputs. As in §2.3.1, let $\mathcal{I}_\delta(F)$ denote the information conveyed by event F drawn from probability distribution δ. Then $\mathcal{I}_{\delta_U}(o)$ quantifies the information contained in o about both the untrusted inputs and the probabilistic choices made by the program, but $\mathcal{I}_{\delta_Y}(o)$ quantifies only the information about the probabilistic choices. And $\mathcal{Q}_{con}$ quantifies just the information about the untrusted inputs:

Corollary 3.1. $\mathcal{Q}_{con}(\mathcal{E}, b'_U) = \mathcal{I}_{\delta_U}(o) - \mathcal{I}_{\delta_Y}(o)$.

Proof. Identical to the proof of theorem 2.2, with the appropriate textual substitutions for agents and security levels. □

3.2 Quantification of Suppression

We now model a *sender* and *receiver*, who communicate through a program. The receiver, by observing the program's outputs, attempts to determine the sender's inputs. For example, the sender might be a database, and the program might construct a web page using queries to the database; the receiver attempts to reconstruct information in the database from the (incomplete) information in the web page. As another example, the program might model a noisy channel;

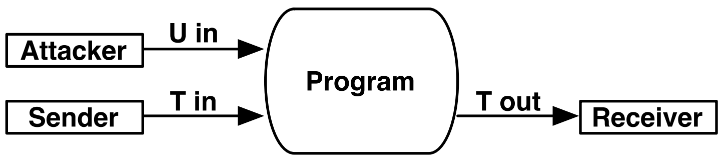

Figure 3.3: Channels in suppression experiment

the sender's inputs are messages, and the receiver attempts to determine what messages were sent.

As with contamination, the program receives trusted inputs as the initial values of variables and produces trusted outputs as the final values of variables. The sender writes the initial values of trusted inputs, and the receiver reads the final values of trusted outputs. These are the only ways that either agent may access any variables. We continue to model an attacker, who can attempt to interfere with the trusted outputs. The attacker writes the initial values of untrusted inputs and may also read the final values of untrusted outputs. The channels between agents and the program are depicted in figure 3.3.

3.2.1 Suppression Experiment Protocol

Formally, a *suppression experiment* $\mathcal{E}$ is described by a tuple,

$$\mathcal{E} = \langle S, b, \sigma_U, \sigma_T \rangle,$$

where S is the program, b is the receiver's prebelief about trusted and untrusted inputs, σ_U is the untrusted projection of the initial state, and σ_T is the trusted projection of the initial state. Note that the receiver's belief concerns the entire initial state because he may not observe any inputs. The protocol for suppression experiments is given in figure 3.4. In the protocol, notation $\sigma \upharpoonright TO$ denotes

A suppression experiment $\mathcal{E} = \langle S, b, \sigma_U, \sigma_T \rangle$ is conducted as follows.

1. The receiver chooses a prebelief b about the trusted and untrusted state.
2. (a) The attacker picks an untrusted state σ_U.
   (b) The sender picks a trusted state σ_T.
3. The receiver predicts the output distribution: $\delta'_R = [\![S]\!]b$.
4. The system executes program S, which produces a state $\sigma' \in \delta'$ as output, where $\delta' = [\![S]\!](\dot{\sigma}_T \otimes \dot{\sigma}_U)$. The receiver observes the trusted projection of the output state: $o = \sigma' \upharpoonright TO$.
5. The receiver infers a postbelief: $b' = (\delta'_R | o)$.

Figure 3.4: Suppression experiment protocol

projection of state σ to trusted outputs. The protocol is a straightforward adaptation of the contamination protocol from §3.1.1.

3.2.2 Suppression Metric

Define the amount of information flow $\mathcal{Q}_{trans}$—that is, the amount of transmission—caused by outcome b' of experiment $\mathcal{E}$ as the improvement in the accuracy of the receiver's belief about trusted inputs:

$$\mathcal{Q}_{trans}(\mathcal{E}, b') \quad \triangleq \quad D(b \upharpoonright TI \rightarrow \dot{\sigma}_T) - D(b' \upharpoonright TI \rightarrow \dot{\sigma}_T),$$

where notation $b \upharpoonright TI$ denotes projection of belief b to trusted inputs.

Quantity $D(b \upharpoonright TI \rightarrow \dot{\sigma}_T)$ is the maximum amount of information the receiver could learn about trusted inputs. Quantity $\mathcal{Q}_{trans}(\mathcal{E}, b')$ is the amount of information the receiver actually learned about trusted inputs from outcome b'. Thus, quantity $D(b' \upharpoonright TI \rightarrow \dot{\sigma}_T)$ is the amount of information the receiver failed to learn about trusted inputs, meaning that it quantifies suppression. Define $\mathcal{S}(\mathcal{E}, b')$ to be that quantity:

$$\mathcal{S}(\mathcal{E}, b') \quad \triangleq \quad D(b' \upharpoonright TI \rightarrow \dot{\sigma}_T).$$

As an example of quantification of suppression, consider the following program:

$$o := i \oplus \mathsf{rnd}()$$

Variables i and o are one-bit input and output variables, respectively. Both variables are trusted. Program expression $\mathsf{rnd}()$ returns a uniformly random bit. Let the receiver have a uniform prebelief b about the value of i. As a result of the suppression experiment protocol, the receiver infers a postbelief b' about i that is uniform, thus $b = b'$. So quantity of transmission $\mathcal{Q}_{trans}$ is 0 bits, and quantity of suppression $\mathcal{S}$ is 1 bit. These quantities are intuitively sensible: the receiver cannot learn anything about i by observing o because of the bit of random noise added by the program.

We can show that $\mathcal{Q}_{trans}$ correctly quantifies the information about trusted input σ_T contained in observation o. Let $\delta_R = ([\![S]\!]b) \upharpoonright TO$ be the receiver's distribution on trusted outputs. Suppose that the sender shares the receiver's belief about untrusted inputs, meaning that the sender's distribution on untrusted inputs is $b|\sigma_T$ when the trusted input is σ_T, and let $\delta_Y = ([\![S]\!](b|\sigma_T)) \upharpoonright TO$ be the sender's distribution on trusted outputs.[5] Then $\mathcal{I}_{\delta_R}(o)$ quantifies the information contained in o about the trusted inputs, untrusted inputs, and the probabilistic choices made by the program. And $\mathcal{I}_{\delta_Y}(o)$ quantifies only the information about the untrusted inputs and the probabilistic choices. Thus $\mathcal{Q}_{trans}$ quantifies just the information about the trusted inputs:

Theorem 3.1. $\mathcal{Q}_{trans}(\mathcal{E}, b') = \mathcal{I}_{\delta_R}(o) - \mathcal{I}_{\delta_Y}(o)$.

Proof. In appendix 3.A. □

[5]Another way to rationalize distribution δ_Y is to recognize that it would be the receiver's distribution on trusted outputs if he were told the value of trusted input σ_T.

Furthermore, a result similar to theorem 2.7 holds for $\mathcal{Q}_{trans}$: if the sender and receiver use the same distribution δ_T on trusted inputs, then expected amount of flow $\mathsf{E}[\mathcal{Q}_{trans}]$ is equal to the mutual information between trusted inputs T_{in} and trusted outputs T_{out}, given the untrusted inputs U_{in}. The expectation is with respect to observation o and distribution δ_T.

Corollary 3.2. *Let $\mathcal{E} = \langle S, (b|\sigma_U), \delta_T, \sigma_U \rangle$, where $b \upharpoonright TI = \delta_T$. Then:*

$$\mathsf{E}[\mathcal{Q}_{trans}(\mathcal{E})] = \mathcal{I}(T_{in}, T_{out}|U_{in}).$$

Proof. The proof is essentially identical to the proof of theorem 2.7, substituting "trusted" for "high" and "untrusted" for "low," T for H and U for L, etc. □

If program S does not mention any untrusted inputs, the conditioning on untrusted input σ_U can be eliminated from corollary 3.2. In this case, the expected amount of flow is simply the mutual information between the trusted inputs and the trusted outputs. This coincides with the standard information-theoretic model of a communication channel [32], in which there are no untrusted inputs—suppression occurs only when random errors are introduced by the channel itself.

Finally, as an example of quantifying both contamination and suppression, consider this program:

$$o := i \oplus u$$

Recall that i and o are one-bit, trusted input and output variables, and that u is a one-bit, untrusted input variable. Suppose the receiver has a uniform prebelief about inputs i and u. Then the quantity of suppression is 1 bit because the receiver cannot learn anything about i. Likewise, if we treat the receiver as a user—allowing him to observe i and o—then the quantity of contamination is 1 bit because he learns everything about u.

3.2.3 Attacker-controlled Suppression

Sometimes the attacker can control how much suppression occurs. For example, consider the following program:

$$o := i + u$$

Assume that inputs i and u are integers in the interval $[1, M]$ for some $M > 1$. Output o is therefore an integer in $[2, 2M]$. If the receiver observes that o is 2, the receiver can infer that $u = i = 1$. Hence the attacker, by choosing $u = 1$, can make it possible that no information about i is suppressed—though not necessary, because i might be set to some integer other than 1. But if the attacker sets u to M, no matter what value of o the receiver observes, all values of i are still possible. Hence the attacker, by choosing $u = M$, can make it possible that all information about i is suppressed. We now formalize this intuition.

Define the quantity of *attacker-controlled suppression* $\mathcal{S}_A$ for a program S, receiver prebelief b, and trusted input σ_T as follows:

$$\begin{aligned}\mathcal{S}_A(S, b, \sigma_T) \quad \triangleq & \quad \max_{\sigma_U, b' \in \mathcal{B}(\langle S, b, \sigma_U, \sigma_T \rangle)} \mathcal{S}(\langle S, b, \sigma_U, \sigma_T \rangle, b') \\ & \quad - \min_{\sigma_U, b' \in \mathcal{B}(\langle S, b, \sigma_U, \sigma_T \rangle)} \mathcal{S}(\langle S, b, \sigma_U, \sigma_T \rangle, b')\end{aligned}$$

This quantity is the difference between the maximum and the minimum amount of suppression possible over any choice of inputs σ_U made by the attacker. For the program above with a uniform receiver prebelief, the quantity of attacker-controlled suppression is $\lg M$ bits. This is intuitively sensible, because the attacker can control whether it is possible for the receiver to learn everything or nothing about i.

Consider this revision of the program we have been considering:

$$o := i1 + i2 + u$$

Assume that $i1$ and $i2$ are integers in the interval $[1, M]$. If the receiver observes that $o = 3$, the receiver can again infer the exact values of $i1$, $i2$, and u. So the attacker can again make it possible that no information is suppressed. But if the attacker sets u to M, then the receiver will observe that o is in $[M + 2, 3M]$. Note that this allows the receiver to eliminate some possibilities for the input values, since they cannot sum to less than $M + 2$. Hence if the receiver's prebelief on the inputs is uniform, his postbelief will not be uniform, meaning that he learned information about the inputs and that some information was not suppressed. For example, suppose that the receiver observes that $o = M + 2$. There are $\binom{M+1}{2} = \frac{M(M+1)}{2}$ ways to choose input values that sum to $M + 2$. Each of these will be equally likely, so the postbelief will assign each probability $\frac{2}{M(M+1)}$. (But the remaining ways to choose inputs—those that do not sum to $M+2$—will have probability 0, establishing that this distribution is not uniform.) The amount of suppression is therefore $\lg \frac{M(M+1)}{2}$, which is always less than the total amount of information the receiver could have learned, $\lg M^2$. This is intuitively sensible, because the attacker can no longer suppress all the information about trusted inputs $i1$ and $i2$.

3.2.4 Program Suppression

Consider the following program:

$$\textbf{if } u \textbf{ then } i2 := i1 \textbf{ else } i2 := i1 \oplus \mathsf{rnd}()$$

Assume that $i1$ is a 2-bit input variable and $i2$ is a 2-bit output variable. If the attacker sets u to **true**, then $i2$ equals $i1$ and is no information is suppressed. But if the attacker sets u to **false**, all information about $i1$ is suppressed. It would be useful to quantify the amount of suppression that the attacker directly controls, versus the amount that is intrinsic in the program itself. The metric for attacker-controlled suppression did not make this distinction.

Toward that goal, define the quantity of *program suppression* $\mathcal{S}_P$ as follows:

$$\mathcal{S}_P(\mathcal{E}, b') \quad \triangleq \quad D((b'|\sigma_U) \upharpoonright TI \twoheadrightarrow \dot{\sigma}_T).$$

This definition differs from the definition of suppression $\mathcal{S}$ only by conditioning receiver postbelief b' on untrusted input σ_U. This conditioning yields the receiver's postbelief were he told the attacker's untrusted inputs. Any remaining suppression must come solely from the program.

Define the quantity of *attacker-controlled program suppression* $\mathcal{S}_{PA}$ for a program S, receiver prebelief b, and trusted input σ_T as follows:

$$\begin{aligned}\mathcal{S}_{PA}(S, b, \sigma_T) \quad \triangleq \quad & \max_{\sigma_U, b' \in \mathcal{B}(\langle S, b, \sigma_U, \sigma_T \rangle)} \mathcal{S}_P(\langle S, b, \sigma_U, \sigma_T \rangle, b') \\ & - \min_{\sigma_U, b' \in \mathcal{B}(\langle S, b, \sigma_U, \sigma_T \rangle)} \mathcal{S}_P(\langle S, b, \sigma_U, \sigma_T \rangle, b')\end{aligned}$$

This quantity is the difference between the maximum and the minimum amount of program suppression possible over any choice of inputs σ_U made by the attacker. For the program above with a uniform receiver prebelief, the quantity of attacker-controlled program suppression is 2 bits. This is intuitively sensible, because the attacker controls whether $i1$ is completely suppressed.

3.3 Error-Correcting Codes

An *error-correcting code* adds redundant information to a message so that suppression can be detected and corrected. One of the simplest error-correcting codes is the *repetition code* R_n [4], which adds redundancy by repeating a message n times to form a *code-word*. For example, R_3 would encode message 1 as code-word 111. The code-word is sent over a noisy channel, which might corrupt the code-word; the receiver receives this possibly corrupted *word* from the channel. For example, the sender might send code-word 111 yet the receiver could receive word 101. To decode the received word, the receiver can employ *nearest-neighbor decoding*: the nearest neighbor of a word w is the[6] code-word c that is closest to w by the *Hamming distance* metric d. Treating words as vectors of symbols, Hamming distance $d(w, x)$ between words w and x is the number of positions i at which $w_i \neq x_i$. For the repetition code, nearest-neighbor decoding is a majority vote: a word is decoded to the symbol that occurs most frequently in the word. For example, word 101 would be decoded to code-word 111, thus to message 1, but 001 would be decoded to message 0.

Consider the following program, which models the *binary symmetric channel* often studied in information theory:[7]

```
$BSC$ :   $i := 1$;
          while $i \leq n$ do
            $v_i := t_i \ {}_p[] \ v_i :=$ not $t_i$;
            $i := i + 1$
```

BSC takes as trusted input an n-bit variable t, and outputs n-bit trusted variable v. Each bit of the input has probability $1 - p$ of being flipped in the output.

If $n = 1$ and the receiver has a uniform prebelief on trusted input t, then after executing BSC and observing v, the receiver's postbelief b' ascribes probability p to an input t such that $t = v$. The amount of program suppression $\mathcal{S}_P$ is thus $-\lg p$. But suppose that the sender and receiver employ repetition code R_3 with program BSC: the sender encodes a one-bit input s into three bits t_1, t_2, t_3

[6]The nearest neighbor is not necessarily unique for some codes, in which case an arbitrary nearest neighbor is chosen.

[7]Recall from chapter 2 that probabilistic choice $S_1 \ {}_p[] \ S_2$, where $0 \leq p \leq 1$, executes program S_1 with probability p or S_2 with probability $1 - p$.

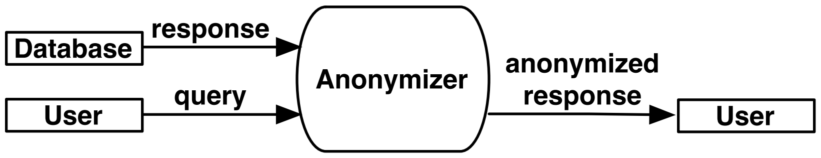

Figure 3.5: Model of anonymizer

(so n = 3), inputs those bits to BSC, then the receiver gets three bits v_1, v_2, v_3 as output, and decodes them to one bit r. Let this composed program be $R_3(BSC)$. Assuming for simplicity that the receiver has a uniform prebelief, postbelief b' ascribes probability $p^3 + 3p^2(1 - p)$ to actual input s.[8] The amount of program suppression $\mathcal{S}_P$ is thus $-\lg(p^3+3p^2(1-p))$. So for any $p > 1/2$ (i.e., for any channel at least slightly biased toward correct transmission), the program suppression from $R_3(BSC)$ is less than the program suppression from BSC. Repetition code R_3 thus corrects program suppression.

3.4 Statistical Databases

The introduction to this chapter suggested that mechanisms used by statistical databases to create anonymized responses to queries can be characterized as sacrificing integrity to improve confidentiality. We can now make this characterization precise by using our models.

As depicted in figure 3.5, we model the anonymizer with a program that receives two inputs. The first input is the user's query, which contains pub-

[8]This probability can be derived either by evaluating the program semantics directly, or by the following argument. Decoded output r equals input s if exactly zero or one bits in code word $t_1t_2t_3$ are flipped during transmission. Each bit t_i is transmitted correctly with probability p and flipped with probability $1 - p$. The probability that zero bits are flipped is thus p^3; the probability that a particular bit t_i is flipped is $p^2(1 - p)$; and there are three possible single bits that could be flipped. So the total probability of correct decoding is $p^3 + 3p^2(1 - p)$.

lic information. The second input is a response containing secret information from the database—perhaps even the entire contents of the database. The anonymizer produces an *anonymized response* as public output.[9] The user is an attacker against confidentiality, because he might be attempting to learn secret information through his query. Since the model we have just described coincides with our model for quantitative confidentiality, it is straightforward to analyze the amount of information leaked by the anonymizer using the techniques in §2.3. In particular, metric $\mathcal{Q}$ is the quantity of leakage.

But the anonymizer also acts as a noisy communication channel, where the database is the sender and the user is the receiver. The input to this channel is trusted input from the database, and the output from the channel is the trusted, anonymized response to the user. The query input from the user could be deemed untrusted, but the user is not an attacker against integrity because he does not attempt to reduce the amount of information he learns through the channel—indeed, he would prefer to increase the amount. So although there is no attacker-controlled suppression, the anonymizer causes program suppression as quantified by $\mathcal{S}_P$.

We can relate the quantity of leakage to the amount of program suppression. Let A be the anonymizer program, b be the user's prebelief about the database, d be the actual database contents, q be the user's query, and b' be the user's postbelief after observing the anonymized response. Then we obtain the following theorem:

[9]The anonymizer might also produce some output about the anonymization it just performed, and this output might be stored in the database and used during future anonymizations. This output would be secret; we do not model it here.

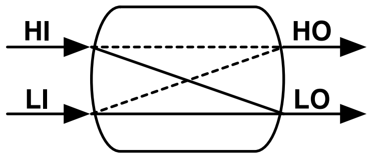

Figure 3.6: Information flows in a system. Dashed lines are uninteresting from our security perspective.

Theorem 3.2. $D(b \rightharpoonup \dot{d}) = \mathcal{Q}(\langle A, b, d, q\rangle, b') + \mathcal{S}_P(\langle A, b, d, q\rangle, b')$.

Proof. In appendix 3.A. □

This theorem means that the quantity of leakage plus the quantity of program suppression is constant for a given experiment and outcome. That constant is inaccuracy $D(b \rightharpoonup \dot{d})$ in the user's prebelief b about the database contents d. This is intuitively sensible, because $D(b \rightharpoonup \dot{d})$ is the total amount of information the user could possibly learn about the database contents. All of that information is either communicated to the user (quantity $\mathcal{Q}(\langle A, b, d, q\rangle, b')$) or suppressed (quantity $\mathcal{S}_P(\langle A, b, d, q\rangle, b')$).

3.5 Duality of Integrity and Confidentiality

Consider a program that processes two levels of information, low and high, denoted L and H. We take as the defining characteristic of low information that its use be unrestricted in the program. For confidentiality, low is therefore synonymous with public, and for integrity, low is synonymous with trusted. Analogously, we take as the defining characteristic of high information that its use be restricted in the program. So for confidentiality, high is synonymous with secret, and for integrity, high is synonymous with untrusted.

	Flow	Attenuation
$HI \rightarrow LO$	C: leakage	C: hiding
	I: contamination	I: hygiene
$LI \rightarrow LO$	C: —	C: —
	I: transmission	I: suppression

Figure 3.7: Dualities between integrity (I) and confidentiality (C)

Let HI denote the high inputs to the system; LO, the low outputs; etc. As depicted in figure 3.6, there are four information flows between inputs and outputs in this system: $LI \rightarrow LO$, $HI \rightarrow LO$, $LI \rightarrow HO$, and $HI \rightarrow HO$. The two flows to HO are uninteresting from our security perspective because high outputs do not need to be protected—that is, for confidentiality, it does not matter what information flows to secret outputs; and for integrity, it does not matter what information flows to untrusted outputs. However, the remaining two flows to LO are interesting and exhibit dualities, which are summarized in figure 3.7 and discussed below.

Flow $HI \rightarrow LO$ is the standard problem with which information-flow security has been concerned. For confidentiality, this is the flow, or *leakage*, from secret inputs to public outputs; §2.2 and §2.3 presented our framework for quantification of leakage. For integrity, this is the flow from untrusted inputs to trusted outputs; this flow was named *contamination* in §3.1. Contamination of trusted information is therefore the information-flow dual of leakage of secret information: both quantify how much information flows between inputs and outputs at different security levels. Indeed, our framework for quantification of contamination was nearly the same as our framework for quantification of leakage. We needed to introduce a new agent, the user, in the integrity model. But the user could have been included in the confidentiality model; the user's role there would have been to choose secret inputs. Note that the user and attacker reverse roles in the two models: for confidentiality, the attacker holds belief about high (secret) inputs, and for integrity, the user holds belief about high (untrusted) inputs.

Define *attenuation* as the amount of information that does not flow from an input to an output. The amount of actual flow of information is therefore the amount of information that could possibly flow less the amount of attenuation. For confidentiality, the attenuation of $HI \rightarrow LO$ is the distance $D(b'_H \rightarrowtriangle \dot{\sigma}_H)$ from attacker's postbelief b'_H to state σ_H. This distance is the amount of secret information that is not leaked to the attacker; we could call this attenuation *hiding* of information. Dually, for integrity, distance $D(b'_U \rightarrowtriangle \dot{\sigma}_U)$ is the amount of untrusted information that does contaminate the trusted outputs; we could call this attenuation *hygiene* because it preserves the "cleanliness" of the trusted outputs.

Flow $LI \rightarrow LO$ can be understood as one of the standard problems with which classical information theory is concerned. For integrity, this is the flow, or transmission, from trusted inputs to trusted outputs; our framework for quantifying the flow and its attenuation, which we named *suppression*, was given in §3.2. For confidentiality, this flow is uninteresting: the amount of information that flows from public inputs to public outputs does not characterize how the program leaks or hides secret information. So there does not seem to be a dual to this flow.

3.6 Related Work

Newsome, Song, and McCamant [94] quantify the amount of influence an attacker can exert over the execution of a program as the logarithm of the size of the set of possible outputs. Assuming that programs are deterministic and that all inputs are either under the control of the attacker or are fixed constants, this quantity is the channel capacity of the program. Our definition of contamination generalizes this definition by allowing probabilistic programs and trusted inputs that are not under the control of the attacker. Also, their definition conservatively assumes a uniform distribution over outputs, but the definitions given here allow arbitrary distributions over inputs and outputs. However, they implement a dynamic analysis that automatically quantifies influence in real-world programs.

Kifer and Gehrke [63] quantify the utility of anonymized data with relative entropy (there called Kullback-Leibler divergence). They use this metric to select among different anonymizations of a dataset.

Biba [15] first identified a duality between confidentiality and integrity, modeling integrity with a dual of the Bell–LaPadula model of confidentiality. Similar dualities have been exploited in Flume [67] and recent versions of Jif [22].

Clark and Wilson [26] propose a different kind of integrity policy, suitable for commercial organizations, based on well-formed transactions and verification procedures. We have not investigated quantitative generalizations of this policy.

3.7 Summary

This chapter presents an information-flow model for quantification of integrity. We introduced two novel information-flow integrity metrics, contamination and suppression. Both metrics are defined by adapting our belief-based model (in chapter 2) for quantification of confidentiality. We have shown that our metric for suppression agrees with the classical information-theoretic metric for channel capacity. We have also applied our definition to the analysis of error-correcting codes and statistical databases.

3.A Appendix: Proofs

Theorem 3.1 $\mathcal{Q}_{trans}(\mathcal{E}, b') = \mathcal{I}_{\delta_R}(o) - \mathcal{I}_{\delta_Y}(o)$.

Proof.

$$\begin{array}{ll}
 & \mathcal{Q}_{trans}(\mathcal{E}, b') \\
= & \quad \langle \text{ Definition of } \mathcal{Q}_{trans} \ \rangle \\
 & D(b \upharpoonright TI \rightarrow \dot{\sigma}_T) - D(b' \upharpoonright TI \rightarrow \dot{\sigma}_T) \\
= & \quad \langle \text{ Definitions of } D \text{ and point mass } \rangle \\
 & -\lg(b \upharpoonright TI)(\sigma_T) + \lg(b' \upharpoonright TI)(\sigma_T) \\
= & \quad \langle \text{ Lemma 3.1 (below), properties of } \lg \ \rangle \\
 & -\lg \Pr_{\delta_R}(o) + \lg \Pr_{\delta_Y}(o) \\
= & \quad \langle \text{ Definition of } \mathcal{I} \ \rangle \\
 & \mathcal{I}_{\delta_R}(o) - \mathcal{I}_{\delta_Y}(o)
\end{array}$$

□

Lemma 3.1. $(b' \upharpoonright TI)(\sigma_T) = (b \upharpoonright TI)(\sigma_T) \cdot \frac{\delta_S(o)}{\delta_R(o)}$.

Proof.

$$\begin{array}{ll}
 & (b' \upharpoonright TI)(\sigma_T) \\
= & \quad \langle \text{ Definition of } b' \text{ in corruption experiment protocol } \rangle \\
 & (((\llbracket S \rrbracket b)|o) \upharpoonright TI)(\sigma_T) \\
= & \quad \langle \text{ Definition of } \delta \upharpoonright TI \ \rangle \\
 & (\sum \sigma \ : \ \sigma \upharpoonright TI = \sigma_T \ : \ ((\llbracket S \rrbracket b)|o)(\sigma))
\end{array}$$

$=\quad \langle$ Definition of $\delta|o$ $\rangle$

$$(\textstyle\sum \sigma \,:\, \sigma \upharpoonright TI = \sigma_T \;\wedge\; \sigma \upharpoonright TO = o \,:\, \dfrac{([\![S]\!]b)(\sigma)}{(([\![S]\!]b) \upharpoonright TO)(o)})$$

$=\quad \langle$ Definition of $\delta \upharpoonright T$ $\rangle$

$$(\textstyle\sum \sigma \,:\, \sigma \upharpoonright T = (\sigma_T \cup o) \,:\, \dfrac{([\![S]\!]b)(\sigma)}{(([\![S]\!]b) \upharpoonright TO)(o)})$$

$=\quad \langle$ Distributivity $\rangle$

$$\frac{1}{(([\![S]\!]b) \upharpoonright TO)(o)} \cdot (\textstyle\sum \sigma \,:\, \sigma \upharpoonright T = (\sigma_T \cup o) \,:\, ([\![S]\!]b)(\sigma))$$

$=\quad \langle$ Definition of δ_R $\rangle$

$$\frac{1}{\delta_R(o)} \cdot (\textstyle\sum \sigma \,:\, \sigma \upharpoonright T = (\sigma_T \cup o) \,:\, ([\![S]\!]b)(\sigma))$$

$=\quad \langle$ Definition of $[\![S]\!]\delta$ $\rangle$

$$\frac{1}{\delta_R(o)} \cdot (\textstyle\sum \sigma \,:\, \sigma \upharpoonright T = (\sigma_T \cup o) \,:\, (\textstyle\sum \sigma' \,:\, b(\sigma') \cdot ([\![S]\!]\sigma')(\sigma)))$$

$=\quad \langle$ Input is immutable, so σ and σ' must agree on it $\rangle$

$$\frac{1}{\delta_R(o)} \cdot (\textstyle\sum \sigma \,:\, \sigma \upharpoonright T = (\sigma_T \cup o) \,:\, \\ (\textstyle\sum \sigma' \,:\, \sigma' \upharpoonright TI = \sigma_T \,:\, b(\sigma') \cdot ([\![S]\!]\sigma')(\sigma)))$$

$=\quad \langle$ Associativity $\rangle$

$$\frac{1}{\delta_R(o)} \cdot (\textstyle\sum \sigma' \,:\, \sigma' \upharpoonright TI = \sigma_T \,:\, \\ (\textstyle\sum \sigma \,:\, \sigma \upharpoonright T = (\sigma_T \cup o) \,:\, b(\sigma') \cdot ([\![S]\!]\sigma')(\sigma)))$$

$=\quad \langle$ Distributivity $\rangle$

$$\frac{1}{\delta_R(o)} \cdot (\textstyle\sum \sigma' \,:\, \sigma' \upharpoonright TI = \sigma_T \,:\, b(\sigma') \\ \cdot (\textstyle\sum \sigma \,:\, \sigma \upharpoonright T = (\sigma_T \cup o) \,:\, ([\![S]\!]\sigma')(\sigma)))$$

$=\quad \langle$ Unit of $\cdot$ $\rangle$

$$\frac{1}{\delta_R(o)} \cdot \frac{(b \restriction TI)(\sigma_T)}{(b \restriction TI)(\sigma_T)} \cdot (\textstyle\sum \sigma' \,:\, \sigma' \restriction TI = \sigma_T \,:\, b(\sigma')$$

$$\cdot\, (\textstyle\sum \sigma \,:\, \sigma \restriction T = (\sigma_T \cup o) \,:\, ([\![S]\!]\sigma')(\sigma)))$$

$=$ $\langle$ Distributivity $\rangle$

$$\frac{1}{\delta_R(o)} \cdot (b \restriction TI)(\sigma_T) \cdot (\textstyle\sum \sigma' \,:\, \sigma' \restriction TI = \sigma_T \,:\, \dfrac{b(\sigma')}{(b \restriction TI)(\sigma_T)}$$

$$\cdot\, (\textstyle\sum \sigma \,:\, \sigma \restriction T = (\sigma_T \cup o) \,:\, ([\![S]\!]\sigma')(\sigma)))$$

$=$ $\langle$ Definition of $b|U$, using range of σ' $\rangle$

$$\frac{1}{\delta_R(o)} \cdot (b \restriction TI)(\sigma_T) \cdot (\textstyle\sum \sigma' \,:\, \sigma' \restriction TI = \sigma_T \,:\, (b|\sigma_T)(\sigma')$$

$$\cdot\, (\textstyle\sum \sigma \,:\, \sigma \restriction T = (\sigma_T \cup o) \,:\, ([\![S]\!]\sigma')(\sigma)))$$

$=$ $\langle$ Distributivity $\rangle$

$$\frac{1}{\delta_R(o)} \cdot (b \restriction TI)(\sigma_T) \cdot (\textstyle\sum \sigma' \,:\, \sigma' \restriction TI = \sigma_T \,:$$

$$(\textstyle\sum \sigma \,:\, \sigma \restriction T = (\sigma_T \cup o) \,:\, (b|\sigma_T)(\sigma') \cdot ([\![S]\!]\sigma')(\sigma)))$$

$=$ $\langle$ Associativity $\rangle$

$$\frac{1}{\delta_R(o)} \cdot (b \restriction TI)(\sigma_T) \cdot (\textstyle\sum \sigma \,:\, \sigma \restriction T = (\sigma_T \cup o) \,:$$

$$(\textstyle\sum \sigma' \,:\, \sigma' \restriction TI = \sigma_T \,:\, (b|\sigma_T)(\sigma') \cdot ([\![S]\!]\sigma')(\sigma)))$$

$=$ $\langle$ Input is immutable, so σ and σ' must agree on it $\rangle$

$$\frac{1}{\delta_R(o)} \cdot (b \restriction TI)(\sigma_T) \cdot (\textstyle\sum \sigma \,:\, \sigma \restriction T = (\sigma_T \cup o) \,:$$

$$(\textstyle\sum \sigma' \,:\, (b|\sigma_T)(\sigma') \cdot ([\![S]\!]\sigma')(\sigma)))$$

$=$ $\langle$ Definition of $[\![S]\!]\delta$ $\rangle$

$$\frac{1}{\delta_R(o)} \cdot (b \restriction TI)(\sigma_T) \cdot (\textstyle\sum \sigma \,:\, \sigma \restriction T = (\sigma_T \cup o) \,:\, ([\![S]\!](b|\sigma_T))(\sigma))$$

$=$ $\langle$ Definition of $b|U$ $\rangle$

$$\frac{1}{\delta_R(o)} \cdot (b \restriction TI)(\sigma_T) \cdot (\sum \sigma \,:\, \sigma \restriction TO = o \,:\, (\llbracket S \rrbracket(b|\sigma_T))(\sigma))$$

$=$ $\langle$ Definition of $\delta \restriction TO$ $\rangle$

$$\frac{1}{\delta_R(o)} \cdot (b \restriction TI)(\sigma_T) \cdot ((\llbracket S \rrbracket(b|\sigma_T)) \restriction TO)(o)$$

$=$ $\langle$ Definition of δ_S $\rangle$

$$\frac{1}{\delta_R(o)} \cdot (b \restriction TI)(\sigma_T) \cdot \delta_S(o)$$

$=$ $\langle$ Commutativity $\rangle$

$$(b \restriction TI)(\sigma_T) \cdot \frac{\delta_S(o)}{\delta_R(o)}$$

$\square$

CHAPTER 4

FORMALIZATION OF SECURITY POLICIES[*]

The Trusted Computer System Evaluation Criteria (TCSEC) [37], also known as the "Orange Book," establishes *verified design* as the highest security certification that a computer system can obtain.[1] To meet (in part) the criteria for verified design, a system must be accompanied by a formal security policy, a formal design, and a formal or informal proof that the design satisfies the policy. Some lower levels of certification also require the statement of a formal security policy.[2] So formal techniques for specification of security policies are necessary to achieve high levels of certification.

Recall from chapter 1 that the theory of trace properties seems appealing as a formal technique for specification of security policies, but that security policies such as noninterference and mean response time are not trace properties. Sets of trace properties, however, are sufficient to formalize security policies. In chapter 1, we named these sets *hyperproperties*. A theory of hyperproperties is developed in this chapter. We generalize safety and liveness, and their topological characterizations, from trace properties to hyperproperties. We identify a subclass of hypersafety, called *k-safety*, for which we give a relatively complete verification methodology. And we show that every hyperproperty is the intersection of a safety hyperproperty and a liveness hyperproperty.

This chapter proceeds as follows. Hyperproperties, hypersafety, k-safety, and hyperliveness are defined and explored in §4.1, §4.2, §4.3, and §4.4, respectively. A topological account of hyperproperties is given in §4.5. The hyperpro-

[*]This chapter contains material from a previously published paper [29], which is © 2008 IEEE and reprinted, with permission, from *Proceedings of the 21st IEEE Computer Security Foundations Symposium*.

[1]Verified design is designated "Class A1" by the TCSEC.

[2]These certifications are *structured protection* and *security domains*, designated "Class B2" and "Class B3" by the TCSEC.

perty intersection theorem is presented in §4.6, and §4.7 concludes. Most proofs are delayed from the main body to appendix 4.A.

4.1 Hyperproperties

We model system execution with traces, where a *trace* is a sequence of states; by employing rich enough notions of state, this model can encode other representations of execution.[3]

The structure of a state is not important in the following definitions, so we leave set Σ of states abstract. However, the structure of a state is important for real examples, and we introduce predicates and functions, on states and on traces, as needed to model events, timing, probability, etc.

Traces may be finite or infinite sequences, which we categorize into sets:

$$\begin{aligned} \Psi_{\mathsf{fin}} &\triangleq \Sigma^*, \\ \Psi_{\mathsf{inf}} &\triangleq \Sigma^\omega, \\ \Psi &\triangleq \Psi_{\mathsf{fin}} \cup \Psi_{\mathsf{inf}}, \end{aligned}$$

where Σ^* denotes the set of all finite sequences over Σ, and Σ^ω denotes the set of all infinite sequences over Σ. For trace $t = s_0 s_1 \ldots$ and index $i \in \mathbb{N}$, we define the following indexing notation:

$$\begin{aligned} t[i] &\triangleq s_i, \\ t[..i] &\triangleq s_0 s_1 \ldots s_i, \\ t[i..] &\triangleq s_i s_{i+1} \ldots \end{aligned}$$

[3]Chapter 5 shows how to model a labeled transition system as a set of traces by including transition labels in states, thereby preserving information about the nondeterministic branching structure of the system. This encoding is also used by chapter 5 to model state machines and probabilistic systems.

We denote concatenation of finite trace t and (finite or infinite) trace t' as tt', and we denote the empty trace as ϵ.

A *system* is modeled by a non-empty set of infinite traces, called its *executions*. If an execution terminates (and thus could be represented by a finite trace), we represent it as an infinite trace by infinitely stuttering the final state in the finite trace.

4.1.1 Trace Properties

A *trace property* is a set of infinite traces [5,70]. The set of all trace properties is

$$\mathsf{Prop} \triangleq \mathcal{P}(\Psi_{\mathsf{inf}}),$$

where $\mathcal{P}$ denotes powerset. A set T of traces satisfies a trace property P, denoted $T \models P$, iff all the traces of T are in P:

$$T \models P \triangleq T \subseteq P.$$

Some security policies are expressible as trace properties. For example, consider the policy "The system may not write to the network after reading from a file." Formally, this is the set of traces

$$\begin{aligned} NRW \triangleq \{t \in \Psi_{\mathsf{inf}} \mid \neg(\exists\, i, j \in \mathbb{N} : i < j \;\wedge\; isFileRead(t[i]) \\ \wedge\; isNetworkWrite(t[j]))\}, \end{aligned} \tag{4.1.1}$$

where $isFileRead$ and $isNetworkWrite$ are state predicates.

Similarly, *access control* is a trace property requiring every operation to be consistent with its requestor's rights:

$$\begin{aligned} AC \triangleq \{t \in \Psi_{\mathsf{inf}} \mid (\forall\, i \in \mathbb{N} : rightsReq(t[i]) \\ \subseteq acm(t[i-1])[subj(t[i]), obj(t[i])])\}. \end{aligned} \tag{4.1.2}$$

Function $acm(s)$ yields the access control matrix in state s. Function $subj(s)$ yields the subject who requested the operation that led to state s, function $obj(s)$ yields the object involved in that operation, and function $rightsReq(s)$ yields the rights required for the operation to be allowed.

As another example, *guaranteed service* is a trace property requiring that every request for service is eventually satisfied:

$$\begin{aligned} GS \;\triangleq\; \{t \in \Psi_{\mathsf{inf}} \mid (\forall\, i \in \mathbb{N} : \mathit{isReq}(t[i]) \\ \implies (\exists\, j > i : \mathit{isRespToReq}(t[j], t[i])))\}. \end{aligned} \quad (4.1.3)$$

Predicate $\mathit{isReq}(s)$ identifies whether a request is initiated in state s, and predicate $\mathit{isRespToReq}(s', s)$ identifies whether state s' completes the response to the request initiated in state s.

4.1.2 Hyperproperties

A *hyperproperty* is a set of sets of infinite traces, or equivalently a set of trace properties. The set of all hyperproperties is

$$\begin{aligned} \mathsf{HP} &\triangleq \mathcal{P}(\mathcal{P}(\Psi_{\mathsf{inf}})) \\ &= \mathcal{P}(\mathsf{Prop}). \end{aligned}$$

The interpretation of a hyperproperty as a security policy is that the hyperproperty is the set of systems allowed by that policy.[4] Each trace property in a hyperproperty is an allowed system, specifying exactly which executions must be possible for that system. Thus a set T of traces satisfies hyperproperty $\boldsymbol{H}$, denoted $T \models \boldsymbol{H}$, iff T is in $\boldsymbol{H}$:

$$T \models \boldsymbol{H} \;\triangleq\; T \in \boldsymbol{H}.$$

[4]The hyperproperty might also contain the empty set of traces, although this set does not correspond to a system.

Note the use of bold face to denote hyperproperties (e.g., $\boldsymbol{H}$) and sans serif to denote sets of trace properties (e.g., Prop). Although a hyperproperty and a set of trace properties are mathematically the same kind of object (a set of sets of traces), they are used differently in formulas, hence the different typography. Sets of hyperproperties are simultaneously bold face and sans serif (e.g., $\boldsymbol{\mathsf{HP}}$).

Given a trace property P, there is a unique hyperproperty denoted $[P]$ that expresses the same policy as P. We call this hyperproperty the *lift* of P. For P and $[P]$ to express the same policy, they must be satisfied by the same sets of traces. Thus we can derive a definition of $[P]$:

$$\begin{aligned}
& (\forall T \in \mathsf{Prop} : T \models P \iff T \models [P]) \\
= \;& (\forall T \in \mathsf{Prop} : T \subseteq P \iff T \in [P]) \\
= \;& [P] = \{T \in \mathsf{Prop} \mid T \subseteq P\} \\
= \;& [P] = \mathcal{P}(P).
\end{aligned}$$

Consequently, the lift of P is the powerset of P:

$$[P] \triangleq \mathcal{P}(P).$$

4.1.3 Hyperproperties in Action

Trace properties are satisfied by traces, whereas hyperproperties are satisfied by sets of traces. This additional level of sets means that hyperproperties can be more expressive than trace properties. We explore this added expressivity with some examples.

Secure information flow. Information-flow security policies express restrictions on what information may be learned by users of a system. Users interact with systems by providing inputs and observing outputs. To model this interaction, define $ev(s)$ as the input or output event, if any, that occurs when a system transitions to state s. Assume that at most one event, input or output, can occur at each transition. For a trace t, extend this notation to $ev(t)$, denoting the sequence of events resulting from application of $ev(\cdot)$ to each state in trace t.[5] Further assume that each user of a system is cleared either at confidentiality level L, representing *low* (public) information, or H, representing *high* (secret) information, and that each event is labeled with one of these confidentiality levels. Define $ev_L(t)$ to be the subsequence of low input and output events contained within $ev(t)$, and $ev_{Hin}(t)$ to be the subsequence of high input events contained within $ev(t)$.

Noninterference, as defined by Goguen and Meseguer [46], requires that commands issued by users holding high clearances be removable without affecting observations of users holding low clearances. Treating commands as inputs and observations as outputs, we model this security policy as a hyperproperty requiring a system to contain, for any trace t, a corresponding trace t' with no high inputs yet with the same low events as t:

$$\begin{aligned} \boldsymbol{GMNI} \;\triangleq\; \{T \in \mathsf{Prop} \mid\ & T \in \boldsymbol{SM} \\ & \wedge\ (\forall\, t \in T : (\exists\, t' \in T : ev_{Hin}(t') = \epsilon \\ & \qquad \wedge\ ev_L(t) = ev_L(t')))\}. \end{aligned} \tag{4.1.4}$$

Conjunct $T \in \boldsymbol{SM}$ expresses the requirement, made by Goguen and Meseguer's formalization, that systems are deterministic state machines (§5.4 defines ***SM*** formally). ***GMNI*** is not a trace property because trace t is allowed only if corresponding trace t' is also allowed.

[5]Depending on the nature of events in the particular system that is being modeled, it might be appropriate for $ev(t)$ to eliminate stuttering of events.

Generalized noninterference [81] extends Goguen and Meseguer's definition of noninterference to handle nondeterministic systems, which are the systems modeled by Prop. McLean [86] reformulates generalized noninterference as a policy requiring a system to contain, for any traces t_1 and t_2, an interleaved trace t_3 whose high inputs are the same as t_1 and whose low events are the same as t_2. This is a hyperproperty:

$$\begin{aligned}\boldsymbol{GNI} \;\triangleq\; &\{T \in \mathsf{Prop} \mid (\forall\, t_1, t_2 \in T : (\exists\, t_3 \in T : \\ &\quad ev_{Hin}(t_3) = ev_{Hin}(t_1) \;\wedge\; ev_L(t_3) = ev_L(t_2)))\}. \end{aligned} \tag{4.1.5}$$

GNI is not a trace property because the presence of any two traces t_1 and t_2 in a system necessitates the presence of a third trace t_3.

Observational determinism [85, 102] requires a system to appear deterministic to a low user. Zdancewic and Myers's [130] definition of observational determinism can be formulated as a hyperproperty:

$$\boldsymbol{OD} \;\triangleq\; \{T \in \mathsf{Prop} \mid (\forall\, t, t' \in T : t[0] =_L t'[0] \implies t \approx_L t')\}. \tag{4.1.6}$$

State equivalence relation $s =_L s'$ holds whenever states s and s' are indistinguishable to a low user, and trace equivalence relation $t \approx_L t'$ holds whenever traces t and t' are indistinguishable to a low user. Zdancewic and Myers define trace equivalence in terms of state equivalence, requiring the sequence of states in each trace to be equivalent up to both stuttering and prefix; equivalence up to prefix makes their definition *termination insensitive*—that is, systems are allowed to leak information via termination channels.[6] ***OD*** is not a trace property because whether some trace is allowed in a system depends on all the other traces of the system.

[6]Zdancewic and Myers also require systems to be race free, hence they weaken trace equivalence to hold for each memory location in a state in isolation, not over all memory locations simultaneously. We omit this requirement for simplicity.

Bisimulation-based definitions of information-flow security policies can also be formulated as hyperproperties,[7] which we demonstrate in chapter 5 with Focardi and Gorrieri's [44] bisimulation nondeducibility on compositions (BNDC) and with Boudol and Castellani's [18] definition of noninterference.

All information-flow security policies we investigated turned out to be hyperproperties, not trace properties. This is suggestive, but any stronger statement about the connection between information flow and hyperproperties would require a formal definition of information-flow policies, and none is universally accepted. Nonetheless, we believe that information flow is intrinsically tied to correlations between (not within) executions. And hyperproperties are sufficiently expressive to formulate such correlations, whereas trace properties are not.

Service level agreements. A *service level agreement* (SLA) specifies acceptable performance of a system. Such specifications commonly use statistics such as

- *mean response time*, the mean time that elapses between a request and a response;
- *time service factor*, the percentage of requests that are serviced within a specified time; and
- *percentage uptime*, the percentage of time during which the system is available to accept and service requests.

These statistics can be used to define policies with respect to individual executions of a system or across all executions of a system. In the former case, the

[7]Since hyperproperties are trace-based, this might at first seem to contradict results, such as Focardi and Gorrieri's [44], stating that bisimulation-based definitions are more expressive than trace-based definitions. However, by employing a richer notion of state [105, §1.3] in traces than Focardi and Gorrieri, hyperproperties are able to express bisimulations.

SLA would be a trace property. For example, the policy "The mean response time *in each execution* is less than 1 second" might not be satisfied by a system if there are executions in which some response times are much greater than 1 second. Yet if these executions are rare, the system might still satisfy the policy "The mean response time *over all executions* is less than 1 second." This latter SLA is not a trace property, but it is a hyperproperty:

$$\boldsymbol{RT} \quad \triangleq \quad \{T \in \mathsf{Prop} \mid \mathit{mean}\left(\bigcup_{t \in T} \mathit{respTimes}(t)\right) \leq 1\}. \tag{4.1.7}$$

Function $\mathit{mean}(X)$ denotes the mean[8] of a set X of real numbers, and $\mathit{respTimes}(t)$ denotes the set of response times (in seconds) from request and response events in trace t. Policies derived from the other SLA statistics above can similarly be expressed as hyperproperties.

4.1.4 Beyond Hyperproperties?

Hyperproperties are able to express security policies that trace properties cannot. So it is natural to ask whether there are security policies that hyperproperties cannot express. We have equated security policies with system properties, and we chose to model systems as trace sets. Every property of trace sets is a hyperproperty, so by definition hyperproperties are expressively complete for our formulations of "system" and "security policy." To find security policies that hyperproperties cannot express (if any exist), we would need to examine alternative notions of systems and security policies. Alternative formulations of systems are discussed in chapter 5, but all the formulations considered there turn out to have encodings as trace sets—thus hyperproperties are complete for

[8]Since X might have infinite cardinality, $\boldsymbol{RT}$ requires a definition of the mean of an infinite set (and, for some sets, this mean does not exist). We omit formalizing such a definition here; one possibility is to use *Cesàro means* [54].

those formulations. We do not know whether other formulations exist that do not have such encodings.

One way to generalize the notion of a security policy is to consider policies on sets of systems—for example, *diversity* [100], which requires the systems all to implement the same functionality but to differ in their implementation details. Any such policy, however, could be modeled as a hyperproperty on a single system that is a product[9] of all the systems in the set. So hyperproperties again seem to be sufficient.

4.1.5 Logic and Hyperproperties

We have not given a logic in which hyperproperties may be expressed. The examples in this chapter require only second-order logic. Although higher-order logic might also be useful to express hyperproperties, higher-order logic is reducible to second-order logic [107, §6.2]. So we believe that second-order logic is sufficient to express all hyperproperties. But we do not know whether the full power of second-order logic is necessary to express hyperproperties of interest. This has ramifications for verification of hyperproperties, because although full second-order logic cannot be effectively and completely axiomatized, fragments of it can be [14, §2.3].[10]

[9]The *product* of systems T_1 and T_2 can be defined as system $\{(t_1[0], t_2[0])(t_1[1], t_1[2]) \ldots \mid t_1 \in T_1 \wedge t_2 \in T_2\}$, comprising traces over pairs of states. Generalizing, the product of a set of n systems comprises traces over n-tuples of states.

[10]It is natural to ask whether we could further reduce second-order logic to first-order. Such a reduction is possible, but only with the Henkin, rather than standard, semantics of second-order logic [14, §4.2]. We do not know which of these semantics should be preferred for hyperproperties. However, there are trace properties, and thus hyperproperties, that we conjecture cannot be expressed in first-order logic—for example, the trace property containing the single trace $pqppqqpppqqq\ldots$, where p and q are states. This suggests that the standard semantics is appropriate.

4.1.6 Refinement and Hyperproperties

Programmers use *stepwise refinement* [1,9,33,40,71,126] to develop, in a series of steps, a program that implements a specification. The programmer starts from the specification. Each successive step creates a more concrete specification, ultimately culminating in a specification sufficiently concrete that a computer can execute it. To prove that the final concrete specification correctly implements the original specification, the programmer argues at each step that the new concrete specification *refines* the previous specification. Specification S_1 refines specification S_2, denoted S_1 $\textsc{ref}$ S_2, iff every behavior permitted by S_1 is also permitted by S_2—that is, the set of behaviors of S_1 is a subset of the set of behaviors of S_2.

Specifications might describe behaviors at different levels of abstraction. For example, a specification might describe behaviors of a queue, but a refinement of that specification might use an array to implement this behavior. Or a specification might describe behaviors using critical sections, but a refinement might implement critical sections with semaphores. So programmers need techniques to relate the behaviors described by specifications. *Abstraction functions* [55,56] and *refinement mappings* [1] have been developed for this purpose; both interpret concrete behaviors as abstract behaviors.

Generalizing from these two techniques, let an *interpretation function* be a function of type $\Psi \to \Psi$. Let IF be any class of interpretation functions that (like abstraction functions and refinement mappings) is closed under composition and contains the identity function id.[11]

[11]Abstraction functions must also preserve data type operations, and refinement mappings must preserve externally visible components up to stuttering. But these restrictions are not relevant to our discussion.

An interpretation function α can be lifted to $\mathsf{Prop} \to \mathsf{Prop}$ by applying α to each trace in a set:

$$\alpha(T) \quad \triangleq \quad \{\alpha(t) \mid t \in T\}.$$

System S *α-satisfies* trace property P, denoted $S \models_\alpha P$, iff $\alpha(S) \models P$. Notation $S \models P$, as we have used it so far, thus means that $S \models_{id} P$.

Trace property P_1 refines P_2 under interpretation α, denoted $P_1 \;\text{REF}_\alpha\; P_2$, iff $\alpha(P_1) \subseteq P_2$. So for trace properties, satisfaction is the same relation as refinement, and subset implies refinement—that is, if C is a subset of A, then C refines A (under interpretation id). This implication is desirable, because it permits refinements that resolve nondeterminism by removing traces from a system.

It is well-known that this kind of refinement does not generally work for security policies.[12] For example, recall system π (chapter 1), which nondeterministically chooses to output 0, 1, or the value of a secret bit h. System π satisfies the specification "The possible output values are independent of the values of secrets," which can be formulated as a hyperproperty. But consider a system π' that always outputs h. System π' does not satisfy the specification and therefore cannot refine π, yet $\pi' \subseteq \pi$. So subset does not imply refinement for hyperproperties as it does for trace properties.

Hyperproperty $\boldsymbol{H_1}$ refines $\boldsymbol{H_2}$ under interpretation α, denoted $\boldsymbol{H_1} \;\text{HREF}_\alpha\; \boldsymbol{H_2}$, iff $\alpha(\boldsymbol{H_1}) \subseteq \boldsymbol{H_2}$, where $\alpha(\boldsymbol{H})$ is defined as $\{\alpha(T) \mid T \in \boldsymbol{H}\}$. A natural relationship that we would expect to hold is

$$(\forall\, S \in \mathsf{Prop}, \boldsymbol{H} \in \mathsf{HP} : S \models \boldsymbol{H} \iff [S] \;\text{HREF}_{id}\; \boldsymbol{H}), \tag{4.1.8}$$

[12]Previous work has identified refinement techniques that are valid for use with certain information-flow security policies [17,79,86].

because satisfaction and refinement intuitively should agree (as they did for trace properties). Straightforward application of definitions shows that (4.1.8) holds iff $\boldsymbol{H}$ is subset closed.

Thus, perhaps unsurprisingly, the set of hyperproperties with which refinement works is the set SSC of subset-closed hyperproperties:

$$\begin{aligned}\mathsf{SSC} \triangleq \{\boldsymbol{H} \in \mathsf{HP} \mid (\forall T \in \mathsf{Prop} : T \in \boldsymbol{H} \\ \implies (\forall T' \in \mathsf{Prop} : T' \subseteq T \implies T' \in \boldsymbol{H}))\}.\end{aligned}$$

The lifted trace properties are, of course, members of SSC. But SSC contains more than just the lifted trace properties. For example, observational determinism $\boldsymbol{OD}$ (4.1.6) is subset closed and therefore a member of SSC, but $\boldsymbol{OD}$ is not a lifted trace property.

4.2 Hypersafety

According to Alpern and Schneider [5], the "bad thing" in a safety property must be both

- *finitely observable*, meaning its occurrence can be detected in finite time, and
- *irremediable*, so its occurrence can never be remediated by future events.

No-read-then-write NRW (4.1.1) and access control AC (4.1.2) are both safety. The bad thing for NRW is a finite trace in which a network write occurs after a file read. This bad thing is finitely observable, because the write can be detected in some finite prefix of the trace, and irremediable, because the network write can never be undone. For AC, the bad thing is similarly a finite trace in which an operation is performed without appropriate rights.

For trace properties, a bad thing is a finite trace that cannot be a prefix of any execution satisfying the safety property. A finite trace t is a *prefix* of a (finite or infinite) trace t', denoted $t \leq t'$, iff $t' = tt''$ for some $t'' \in \Psi$.

Definition 4.1. A trace property S is a *safety property* [5] iff

$$(\forall\, t \in \Psi_{\mathsf{inf}} : t \notin S \implies (\exists\, m \in \Psi_{\mathsf{fin}} : m \leq t \wedge (\forall\, t' \in \Psi_{\mathsf{inf}} : m \leq t' \implies t' \notin S))).$$

Define SP to be the set of all safety properties; note that SP is itself a hyperproperty.

We generalize safety to hypersafety by generalizing the bad thing from a finite trace to a finite[13] set of finite traces. Define Obs to be the set of such *observations*:

$$\mathsf{Obs} \triangleq \mathcal{P}^{\mathit{fin}}(\Psi_{\mathsf{fin}}),$$

where $\mathcal{P}^{\mathit{fin}}(X)$ denotes the set of all finite subsets of set X. Prefix $\leq$ on sets of traces is defined as follows:[14]

$$T \leq T' \triangleq (\forall\, t \in T : (\exists\, t' \in T' : t \leq t')).$$

Note that this definition allows T' to contain traces that have no prefix in T.

Definition 4.2. A hyperproperty $\boldsymbol{S}$ is a *safety hyperproperty* (is *hypersafety*) iff

$$(\forall\, T \in \mathsf{Prop} : T \notin \boldsymbol{S} \implies (\exists\, M \in \mathsf{Obs} : M \leq T \wedge (\forall\, T' \in \mathsf{Prop} : M \leq T' \implies T' \notin \boldsymbol{S}))).$$

[13]Infinite sets might seem to be an attractive alternative, and many of the results in the rest of this chapter would still hold. However, the topological characterization given in §4.5 (specifically, propositions 4.4 and 4.5) would be sacrificed.

[14]Other definitions of trace set prefix are possible, but inconsistent with our notion of observation. We discuss this in §4.5.

The definition of hypersafety parallels the definition of safety, but the domains involved now include an extra level of sets. Define SHP to be the set of all safety hyperproperties.

Observational determinism ***OD*** (4.1.6) is hypersafety. The bad thing is a pair of traces that are not low-equivalent despite having low-equivalent initial states. But set SP of all safety properties is not hypersafety: there is no bad thing that prevents an arbitrary trace property from being extended to a safety property.

Safety properties lift to safety hyperproperties:

Proposition 4.1. $(\forall\, S \in \mathsf{Prop} : S \in \mathsf{SP} \iff [S] \in \mathsf{SHP})$.

Proof. In appendix 4.A. □

Refinement of hypersafety. Stepwise refinement works with all safety hyperproperties, because safety hyperproperties are subset closed (c.f. §4.1.6), as stated by the following theorem.

Theorem 4.1. $\mathsf{SHP} \subset \mathsf{SSC}$.

Proof. In appendix 4.A. □

A consequence of theorem 4.1 is that any hyperproperty that is not subset closed cannot be hypersafety. For example, generalized noninterference ***GNI*** (4.1.5) is not subset closed: a system containing traces t_1 and t_2 and interleaved trace t_3 might satisfy ***GNI***, but the subset containing only t_1 and t_2 would not satisfy ***GNI***. Thus ***GNI*** cannot be hypersafety.

4.3 Beyond 2-Safety

Safety properties enjoy a relatively complete verification methodology based on invariance arguments [6]. Although we have not obtained such a methodology for hypersafety, we can use invariance arguments to verify a class of safety hyperproperties by generalizing recent work on verification of secure information flow.

Recall that secure information flow is a hyperproperty but not a trace property. Recent work gives system transformations that reduce verifying secure information flow[15] to verifying a safety property of some transformed system: Pottier and Simonet [99] develop a type system for verifying secure information flow based on simultaneous reasoning about two executions of a program. Darvas et al. [34] show that secure information flow can be expressed in dynamic logic. Barthe et al. [12] give an equivalent formulation for Hoare logic and temporal logic, based on a self-composition construction.

Define the *sequential self-composition of* P as the program $P; P'$, where P' denotes program P, but with every variable renamed to a fresh, primed variable—for example, variable x is renamed to x'. One way to verify that P exhibits secure information flow is to establish the following trace property of transformed program $P; P'$:

> If for every low variable l, before execution $l = l'$ holds, then when execution terminates $l = l'$ still holds, no matter what the values of high variables were.

[15]These reductions are possible because the particular formulations of secure information flow used in each work are actually hypersafety. A formulation that is hyperliveness—which would include all possibilistic information-flow policies, as discussed in §4.4—would not be amenable to these reductions.

Barthe et al. generalize the self-composition operator from sequential composition to any operator that satisfies certain conditions, and they note that parallel composition satisfies these conditions. They also relax the equality constraints in the above trace property to partial equivalence relations. Terauchi and Aiken [115] further generalize the applicability of self-composition by showing that it can be used to verify any *2-safety* property, which they define informally as a "property that can be refuted by observing two finite traces."

Using hyperproperties, we can show that the above results are special cases of a more general theorem. Define a k-safety hyperproperty as a safety hyperproperty in which the bad thing never involves more than k traces:

Definition 4.3. A hyperproperty $\boldsymbol{S}$ is a *k-safety hyperproperty* (is *k-safety*) iff

$$\begin{aligned}(\forall T \in \mathsf{Prop} : T \notin \boldsymbol{S} \implies (\exists M \in \mathsf{Obs} : M \leq T \wedge |M| \leq k \\ \wedge\ (\forall T' \in \mathsf{Prop} : M \leq T' \implies T' \notin \boldsymbol{S}))).\end{aligned}$$

This is just the definition of hypersafety with an added conjunct "$|M| \leq k$". For a given k, define **KSHP(k)** to be the set of all k-safety hyperproperties.

As an example of a k-safety hyperproperty for any k, consider a system that stores a secret by splitting it into k shares. Suppose that an action of the system is to output a share. Then a hyperproperty of interest might be that the system cannot, across all of its executions, output all k shares (thereby outputting sufficient information for the secret to be reconstructed). We denote this k-safety hyperproperty as $\boldsymbol{SecS}_k$.

The 1-safety hyperproperties are the lifted safety properties—that is,

$$\mathsf{KSHP}(1) \ = \ \{[S] \mid S \in \mathsf{SP}\}$$

—since the bad thing for a safety property is a single trace. Thus "1-safety" and "safety" are synonymous.

The Terauchi and Aiken definition of 2-safety properties is limited to deterministic programs that are expressed in a relational model of execution (which we address further in §5.2), and it ignores nonterminating traces. So their 2-safety properties are a strict subset of the 2-safety hyperproperties, $\mathsf{KSHP}(2)$. For example, observational determinism $\boldsymbol{OD}$ (4.1.6) is not a 2-safety property, but it is a 2-safety hyperproperty.

Define the *parallel self-composition of system* S as the product system $S \times S$ consisting of traces over $\Sigma \times \Sigma$:

$$S \times S \triangleq \{(t[0], t'[0])(t[1], t'[1]) \cdots \mid t \in S \wedge t' \in S\}.$$

Define the *k-product* of S, denoted S^k, to be the k-fold parallel self-composition of S, comprising traces over Σ^k. Self-composition $S \times S$ is equivalent to 2-product S^2.

Previous work has shown how to reduce a particular formulation of noninterference of system S to a related safety property of S^2 [12], and how to reduce any 2-safety hyperproperty of S to a related safety property of $S; S'$ [115]. The following theorem generalizes those results. Let Sys be the set of all systems. For any system S, any k-safety hyperproperty $\boldsymbol{K}$ of S can be reduced to a safety property K of S^k, and the proof of the following theorem (in appendix 4.A) shows how to construct K from $\boldsymbol{K}$:

Theorem 4.2. $(\forall\, S \in \mathsf{Sys}, \boldsymbol{K} \in \mathsf{KSHP}(k) : (\exists\, K \in \mathsf{SP} : S \models \boldsymbol{K} \iff S^k \models K))$.

Proof. In appendix 4.A. □

Theorem 4.2 provides a verification technique for k-safety: reduce a k-safety hyperproperty to a safety property, then verify that the safety property is satisfied by S^k using an invariance argument. Since invariance arguments are relatively complete for safety properties [6], this methodology is relatively complete for k-safety.

However, theorem 4.2 does not provide the relatively complete verification procedure we seek for hypersafety, because there are safety hyperproperties that are not k-safety for any k. For example, consider the hyperproperty "for any k, a system cannot output all k shares of a secret from a k-secret sharing":

$$\boldsymbol{SecS} \triangleq \bigcup_{k} \boldsymbol{SecS}_k. \tag{4.3.1}$$

SecS is not k-safety for any k. Yet it is hypersafety, since any trace property not contained in it violates some $\boldsymbol{SecS}_k$.

4.4 Hyperliveness

Alpern and Schneider [5] characterize the "good thing" in a liveness property as

- *always possible*, no matter what has occurred so far, and
- *possibly infinite*, so it need not be a discrete event.

For example, guaranteed service GS (4.1.3) is a liveness property in which the good thing is the eventual response to a request. This good thing is always possible, because a state in which a response is produced can always be appended to any finite trace containing a request. And this good thing is not infinite because the response is a discrete event, but *starvation freedom*, which stipulates that a system makes progress infinitely often, is an example of a liveness property with an infinite good thing.

Formally, a good thing is an infinite suffix of a finite trace:

Definition 4.4. Trace property L is a *liveness property* [5] iff

$$(\forall\, t \in \Psi_{\mathsf{fin}} : (\exists\, t' \in \Psi_{\mathsf{inf}} : t \leq t' \ \wedge\ t' \in L)).$$

Define LP to be the set of all liveness properties. Not surprisingly, LP is a hyperproperty.

Just as with hypersafety, we generalize liveness to hyperliveness by generalizing a finite trace to a finite set of finite traces. The definition of hyperliveness is essentially the same as the definition of liveness, except for an additional level of sets:

Definition 4.5. Hyperproperty $\boldsymbol{L}$ is a *liveness hyperproperty* (is *hyperliveness*) iff

$$(\forall\, T \in \mathsf{Obs} : (\exists\, T' \in \mathsf{Prop} : T \leq T' \ \wedge\ T' \in \boldsymbol{L})).$$

Define $\mathsf{\textbf{LHP}}$ to be the set of all liveness hyperproperties.

Mean response time $\boldsymbol{RT}$ (4.1.7) is not liveness but it is hyperliveness: the good thing is that the mean response time is low enough. Given any observation T with any mean response time, it is always possible to extend T, such that the resulting system has a low enough mean response time, by adding a trace that has many quick responses. Note that if this policy were approximated by limiting the maximum response time in each execution, the resulting hyperproperty would be a lifted safety property.

Set LP of all liveness properties is a liveness hyperproperty: every observation can be extended to any liveness property. Similarly, set SP of all safety properties is a liveness hyperproperty: every observation can be extended to a safety property (whose bad thing is "not beginning execution with one of the finite traces in the observation").

The only hyperproperty that is both hypersafety and hyperliveness is ***true***, defined as Prop. The hyperproperty ***false***, defined as $\{\emptyset\}$, is hypersafety but not hyperliveness.[16]

Liveness properties lift to liveness hyperproperties:

Proposition 4.2. $(\forall\, L \in \mathsf{Prop} : L \in \mathsf{LP} \iff [L] \in \mathbf{LHP})$.

Proof. In appendix 4.A. □

Possibilistic information flow. Some information-flow security policies, such as observational determinism ***OD*** (4.1.6), restrict nondeterminism of a system from being publicly observable. However, observable nondeterminism might be useful, for a couple of reasons. First, systems might exhibit nondeterminism due to scheduling. If the scheduler cannot be influenced by secret information (i.e., the scheduler does not serve as a covert timing channel), it is reasonable to allow the scheduler to behave nondeterministically. Second, nondeterminism is a useful modeling abstraction when dealing with probabilistic systems (which we consider in more detail in §5.5). When the exact probabilities for a system are unknown, they can be abstracted by nondeterminism. For at least these reasons, there is a history of research on *possibilistic* information-flow security policies, beginning with nondeducibility [113] and generalized noninterference [81]. Such policies are founded on the intuition that low observers of a system should gain little from their observations. Typically, these policies require that every low observation is consistent with some large set of possible high behaviors.

[16]The false property is the empty set of traces, so it might seem reasonable to define ***false*** as the empty set of trace sets. But then the lift of the false property would not equal ***false***. Note that ***false*** is not satisfied by any system because, by definition, $\emptyset$ is not a system.

McLean [86] shows that possibilistic information-flow policies can be expressed as trace sets that are closed with respect to *selective interleaving functions*. Such functions, given two executions of a system, specify another trace that must also be an execution of the system—as did the definition of generalized noninterference ***GNI*** (4.1.5). Mantel [78] generalizes from these functions to *closure operators*, which extend a set S of executions to a set S' such that $S \subseteq S'$. Mantel argues that every possibilistic information-flow policy can be expressed as a closure operator.

Given a closure operator Cl that expresses a possibilistic information-flow policy, the hyperproperty $\boldsymbol{P}_{Cl}$ induced by Cl is

$$\boldsymbol{P}_{Cl} \triangleq \{Cl(T) \mid T \in \mathsf{Prop}\}.$$

Define the set PIF of all such hyperproperties to be $\bigcup_{Cl} \boldsymbol{P}_{Cl}$. It is now easy to see that these are liveness hyperproperties: any observation T can be extended to its closure.

Theorem 4.3. $\mathsf{PIF} \subset \mathsf{LHP}$.

Proof. In appendix 4.A. $\square$

Possibilistic information-flow policies are therefore never hypersafety.[17]

Temporal logics. Consider the hyperproperty "For every initial state, there is some terminating trace, but not all traces must terminate," denoted as ***NNT***. In branching-time temporal logic, ***NNT*** could be expressed as

$$\Diamond\ terminates, \tag{4.4.1}$$

[17] Another way to reach this conclusion is to observe that closure operators need not yield hyperproperties that are subset closed—yet, by theorem 4.1, every safety hyperproperty is subset closed.

where $terminates$ is a state predicate and $\Diamond$ is the "not never" operator.[18] There is no linear-time temporal predicate that expresses ***NNT***, nor is there a liveness property equivalent to ***NNT*** [69]; an approximation would be a linear-time predicate, or a liveness property, that requires every trace to terminate. However, ***NNT*** is hyperliveness because any finite trace can be extended to a set of executions such that at least one execution terminates.

This example suggests a relationship between hyperproperties and branching-time temporal predicates, and between trace properties and linear-time temporal predicates. We can make this relationship precise by examining the semantics of temporal logic. In both branching time and linear time, a semantic model contains a set of states and a valuation function assigning a Boolean value to each atomic proposition in each state. Additionally, a branching-time model requires a current state and a set of traces, whereas a linear-time model requires a single trace [41]. These requirements differ because a linear-time predicate is a property of a trace, whereas a branching-time predicate is a property of a state and all the future traces that could proceed from that state. Thus, trace properties model linear-time predicates, and hyperproperties model branching-time predicates for a given state.

Moreover, hyperproperties can express policies that branching-time predicates cannot. Consider the trace property "Every trace must end with an infinite number of $good$ states," denoted SAG, where $good$ is a state predicate. In linear-time temporal logic, SAG could be expressed as

$$\rightsquigarrow \Box\, good, \tag{4.4.2}$$

[18]Temporal logics CTL [27] would express this formula as $\mathsf{E}\ \mathsf{F}\ terminates$.

where $\leadsto$ is the "sometime" operator and $\Box$ is the "always" operator. SAG is liveness and thus hyperliveness, but there is no branching-time predicate that expresses it [69].

4.5 Topology

Topology enables an elegant characterization of the structure of hyperproperties, just as it did for trace properties. We begin by summarizing the topology of trace properties [110].

Consider an *observer* of an execution of a system, who is permitted to see each new state as it is produced by the system; otherwise, the system is a black box to the observer. The observer attempts to determine whether trace property P holds of the system. At any point in time, the observer has seen only a finite prefix of the (infinite) execution. Thus, the observer should declare that the system satisfies P, after observing finite trace t, only if all possible extensions of t will also satisfy P. Abramsky names such properties *observable* [3].

Like the bad thing for a safety property, a observable property must be detectable in finite time; and once detected, hold thereafter. Formally, O is a observable property iff

$$(\forall t \in \Psi_{\mathsf{inf}} : t \in O \implies (\exists m \in \Psi_{\mathsf{fin}} : m \leq t \\ \land (\forall t' \in \Psi_{\mathsf{inf}} : m \leq t' \implies t' \in O))).$$

Define $\mathcal{O}$ to be the set of observable properties. This set satisfies two closure conditions. First, if $O_1, \ldots, O_n$ are observable, then $\bigcap_{i=1}^{n} O_i$ is also observable. Second, if $\boldsymbol{O}$ is a (potentially infinite) set of observable properties, then $\bigcup_{O \in \boldsymbol{O}} O$ is also observable. Thus $\mathcal{O}$ is *closed under finite intersections and infinite unions*.

A *topology* on a set S is a set $\mathcal{T} \subseteq \mathcal{P}(S)$ such that $\mathcal{T}$ is closed under finite intersections and infinite unions. Because $\mathcal{O}$ is so closed, it is a topology on Ψ_{inf}. We name $\mathcal{O}$ the *Plotkin* topology, because Plotkin proposed its use in characterizing safety and liveness [5].[19]

The elements of a topology $\mathcal{T}$ are called its *open sets*. A convenient way to characterize the open sets of a topology is in terms of a base or a subbase. A *base* of topology $\mathcal{T}$ is a set $\mathcal{B} \subseteq \mathcal{T}$ such that every open set is a (potentially infinite) union of elements of $\mathcal{B}$. A *subbase* is a set $\mathcal{A} \subseteq \mathcal{T}$ such that the collection of finite intersections of $\mathcal{A}$ is a base for $\mathcal{T}$. The set

$$\mathcal{O}^B \quad \triangleq \quad \{\uparrow t \mid t \in \Psi_{\mathsf{fin}}\}$$

is a base (and a subbase) of the Plotkin topology, where

$$\uparrow t \quad \triangleq \quad \{t' \in \Psi_{\mathsf{inf}} \mid t \leq t'\}$$

is the *completion* of a finite trace t. When $t \leq t'$ we say that t' *extends* t. The completion of t is thus the set of all infinite extensions of t.

Alpern and Schneider [5] noted that, in the Plotkin topology, safety properties correspond to closed sets and liveness properties correspond to dense sets. A *closed* set is the complement (with respect to S) of an open set. If a trace t is not a member of a closed set C, there is some bad thing (specifically, the prefix m of t in the definition of observable as instantiated on open set $\overline{C}$, the complement of C) that is to blame; the existence of such bad things makes C a safety property. Likewise, a set that is *dense* intersects every non-empty open set in $\mathcal{T}$. So for any finite trace t and dense set D, the intersection of $\uparrow t$ (which is open because it is a member of $\mathcal{O}^B$) and D is nonempty. Since any finite trace can be extended to be in D, it holds that D is a liveness property.

[19]Topology $\mathcal{O}$ is also the Scott topology on the ω-algebraic CPO of traces ordered by $\leq$ [110].

We want to construct a topology on sets of traces that extends this correspondence to hyperproperties. The most important step is generalizing the notion of finite observability from trace properties to hyperproperties. In fact, this generalization was already accomplished in §4.2, where a bad thing was generalized from a finite trace to a finite set of finite traces—that is, an observation. The observer, as before, sees the system produce each new state in the execution. However, the observer may now reset the system at any time, causing it to begin a new execution. At any finite point in time, the observer has now collected a finite set of finite (thus partial) executions. An observation is thus an element of Obs, as defined in §4.2.

An extension of an observation should allow the observer to perform additional resets of the system, yielding a larger set of traces. An extension should also allow each execution to proceed longer, yielding longer traces. So extension corresponds to trace set prefix $\leq$ (c.f. §4.2). The completion of observation M is

$$\uparrow M \quad \triangleq \quad \{T \in \mathsf{Prop} \mid M \leq T\}.$$

We can now define our topology on sets of traces in terms of its subbase:

$$\boldsymbol{\mathcal{O}}^{SB} \quad \triangleq \quad \{\uparrow M \mid M \in \mathsf{Obs}\}.$$

The base $\boldsymbol{\mathcal{O}}^{B}$ of our topology is then $\boldsymbol{\mathcal{O}}^{SB}$ closed under finite intersections. The base and subbase turn out to be the same sets:

Proposition 4.3. $\boldsymbol{\mathcal{O}}^{B} = \boldsymbol{\mathcal{O}}^{SB}$.

Proof. In appendix 4.A. □

Finally, our topology $\boldsymbol{\mathcal{O}}$ is $\boldsymbol{\mathcal{O}}^{B}$ closed under infinite unions.

Define $\boldsymbol{\mathcal{C}}$ to be the closed sets in our topology and $\boldsymbol{\mathcal{D}}$ to be the dense sets. Just as safety and liveness correspond to closed and dense sets in the Plotkin topology, hypersafety and hyperliveness correspond to closed and dense sets in our generalization of that topology:

Proposition 4.4. $\mathsf{SHP} = \mathcal{C}$.

Proof. In appendix 4.A. $\square$

Proposition 4.5. $\mathsf{LHP} = \mathcal{D}$.

Proof. In appendix 4.A. $\square$

Our topology $\mathcal{O}$ is actually equivalent to well-known topology. The *Vietoris* (or *finite* or *convex Vietoris*) topology is a standard construction of a topology on sets out of an underlying topology [87, 116]. Our underlying topology was on traces, and we constructed a topology on sets of traces. The Vietoris construction can be decomposed into the *lower Vietoris* and *upper Vietoris* constructions [109], which also yield topologies. Let $\mathfrak{V}_L(\mathcal{T})$ denote the lower Vietoris construction, which given underlying topology $\mathcal{T}$ on space $\mathcal{X}$ produces the topology on $\mathcal{P}(\mathcal{X})$ induced by subbase $\mathfrak{V}_L^{SB}(\mathcal{T})$:

$$\mathfrak{V}_L^{SB}(\mathcal{T}) \triangleq \{\langle O \rangle \mid O \in \mathcal{T}\},$$

where $\langle T \rangle$ is defined[20] as

$$\langle T \rangle \triangleq \{U \in \mathcal{P}(\mathcal{X}) \mid U \cap T \neq \emptyset\}.$$

[20]Operators $[\cdot]$ (from §4.1) and $\langle \cdot \rangle$ are similar to modal logic operators $\Box$ (necessity) and $\Diamond$ (possibility): For trace property T, lift $[T]$ denotes the set of all refinements of T—that is, the hyperproperty in which T is necessary. Similarly, $\langle T \rangle$ denotes the set of all trace properties that share a trace with T—that is, the hyperproperty in which T is always possible.

The following theorem states that our topology is equivalent to the lower Vietoris construction applied to the Plotkin topology:

Theorem 4.4. $\boldsymbol{\mathcal{O}} = \mathfrak{V}_L(\mathcal{O})$.

Proof. In appendix 4.A. □

Smyth [109] established that the lower Vietoris topology is equivalent to the *lower* (or *Hoare*) *powerdomain*, which is a construction used to model the semantics of nondeterminism [98]. So our topology embodies the same intuition about nondeterminism as the lower powerdomain does.

The proof of theorem 4.4 yields another topological characterization of safety hyperproperties: the set of lifted safety properties, closed under infinite intersections and finite unions (denoted as closure operator Cl_C, because these closure conditions characterize a *topology of* <u>*C*</u>*losed sets*), is the set of safety hyperproperties.

Proposition 4.6. $\mathsf{SHP} = Cl_C(\{[S] \mid S \in \mathsf{SP}\})$.

Proof. In appendix 4.A. □

Defining trace set prefix. Recall that trace set prefix $\leq$ is defined as follows:

$$T \leq T' \quad \triangleq \quad (\forall\, t \in T \,:\, (\exists\, t' \in T' \,:\, t \leq t')).$$

For clarity, we use $\leq_L$ instead of $\leq$ to refer to that definition throughout the rest of this section (L stands for <u>L</u>ower Vietoris).

Two natural alternatives to $\leq_L$ are

$$\begin{aligned} T \leq_U T' \quad &\triangleq \quad (\forall\, t' \in T' \,:\, (\exists\, t \in T \,:\, t \leq t')), \\ T \leq_C T' \quad &\triangleq \quad T \leq_L T' \wedge T \leq_U T'. \end{aligned}$$

(U and C stand for Upper and Convex Vietoris. These prefix relations correspond to the eponymous topologies.) However, both alternatives turn out to be unsuitable for our purposes, because they do not correspond to our intuition about finite observability—as we now explain.

Hyperproperty $\boldsymbol{O}$ is observable iff

$$(\forall T \in \mathsf{Prop} : T \in \boldsymbol{O} \implies (\exists M \in \mathsf{Obs} : M \leq T \\ \wedge \ (\forall T' \in \mathsf{Prop} : M \leq T' \implies T' \in \boldsymbol{O}))).$$

Consider using $\leq_U$ for trace set prefix $\leq$. For a concrete example, suppose that $\Sigma = \{a, b, c\}$, $\boldsymbol{O}$ is observable, $T \in \boldsymbol{O}$, and $M = \{a, b\}$. Any T' such that $M \leq_U T'$ must be a member of $\boldsymbol{O}$. Every trace t' in T' must begin with either a or b and cannot begin with c. In particular, T' might contain traces beginning only with b, never with a. Observation M therefore characterizes a system in which a nondeterministic choice to produce c as the first state is not possible. So with $\leq_U$, an observation records what nondeterminism is denied, and all future extensions of that observation are also required to deny that nondeterminism.

In contrast, with $\leq_L$ (i.e., our topology), an observation records what nondeterminism has so far been permitted, and all future extensions of that observation are required also to permit that nondeterminism. Our intuition is that observers of a black-box system can observe permitted nondeterminism (by observing states produced by the system) but not denied nondeterminism. The definition of $\leq_U$ does not correspond to that intuition, but the definition of $\leq_L$ does. Similarly, using $\leq_C$ for trace set prefix leads to observations that record both permitted and denied nondeterminism (because $\leq_C$ is the conjunction of $\leq_L$ and $\leq_U$), and therefore $\leq_C$ does not correspond to our intuition, either.

So neither the upper nor the convex Vietoris topology enjoys open sets that are the observable hyperproperties; consequently, the equivalence of closed sets and hypersafety is lost. Nonetheless, these topologies might be useful for other purposes—for example, in refusal semantics for CSP [57].

4.6 Beyond Hypersafety and Hyperliveness

Security policies can exhibit features of both safety and liveness. For example, consider a policy on a medical information system that must maintain the confidentiality of patient records and must also eventually notify patients whenever their records are accessed [8]. If the confidentiality requirement is interpreted as observational determinism ***OD*** (4.1.6), this system must both prevent bad things (***OD***, which is hypersafety) as well as guarantee good things (eventual notification, which can be formulated as liveness). As another example, consider an asynchronous proactive secret-sharing system [132] that must maintain and periodically refresh a secret. Each share refresh must complete during a given time interval with high probability. Maintaining the confidentiality of the secret can be formulated as ***SecS*** (4.3.1), which is hypersafety. The eventual refresh of the secret shares can be formulated as liveness: every execution eventually completes the refresh if enough servers remain uncompromised. And the high probability that the refresh succeeds within a given time interval is hyperliveness—similar to mean response time ***RT*** (4.1.7). Both of these examples illustrate hyperproperties that are intersections of (hyper)safety and (hyper)liveness.

In fact, as stated by the following theorem, every hyperproperty is the intersection of a safety hyperproperty with a liveness hyperproperty. This theorem

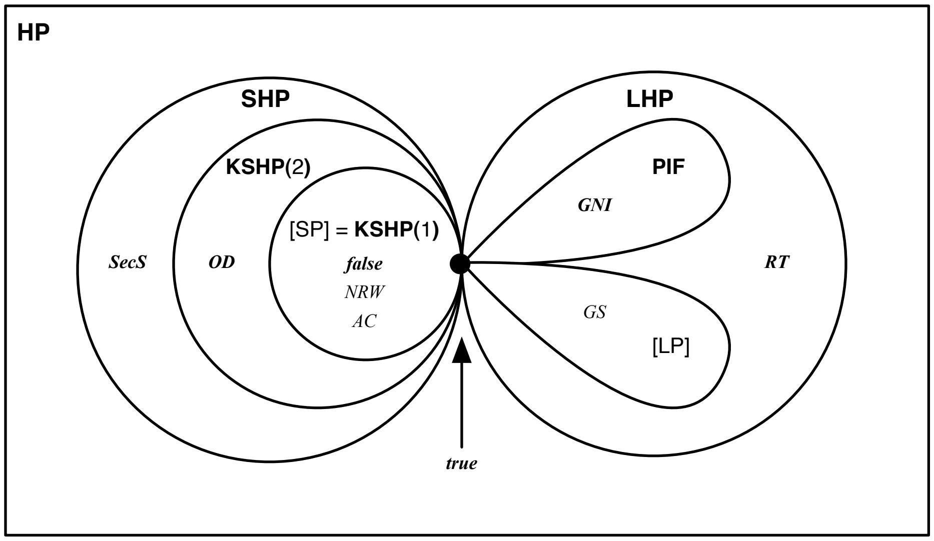

Figure 4.1: Classification of security policies

generalizes the result of Alpern and Schneider [5] that every trace property is the intersection of a safety property and a liveness property:

Theorem 4.5. $(\forall\, \boldsymbol{P} \in \mathsf{HP} : (\exists\, \boldsymbol{S} \in \mathsf{SHP}, \boldsymbol{L} \in \mathsf{LHP} : \boldsymbol{P} = \boldsymbol{S} \cap \boldsymbol{L}))$.

Proof. In appendix 4.A. □

4.7 Summary

This chapter has classified several security policies with hypersafety and hyperliveness. Figure 4.1 summarizes this classification.

We have introduced hyperproperties, which are sets of trace properties and can express security policies that trace properties cannot, such as secure information flow and service level agreements. We have generalized safety and liveness to hyperproperties, showing that every hyperproperty is the intersection of a safety hyperproperty and a liveness hyperproperty. We have also generalized the topological characterization of safety and liveness from trace properties to hyperproperties. We have shown that refinement is applicable with safety hyperproperties.

We have given a relatively complete verification methodology for k-safety hyperproperties that generalizes prior techniques for verifying secure information flow. But we do not know whether there is a relatively complete methodology for all hyperproperties, or even all safety hyperproperties.[21] If such a methodology could be found, security might take its place as "just another" functional requirement to be verified.

[21]If the full power of second-order logic is necessary to express hyperproperties (as discussed at the end of §4.1), such methods could not exist. Nonetheless, methods for verifying fragments of the logic might suffice for verifying hyperproperties that correspond to security policies.

4.A Appendix: Proofs

Bueno and Clarkson [20] have formally verified propositions 4.1 and 4.2, theorems 4.2, 4.3, and 4.5, and an analogue of theorem 4.1 using the Isabelle/HOL proof assistant [95]. We believe that the remaining proofs could also be formally verified.

Proposition 4.1. $(\forall S \in \mathsf{Prop} : S \in \mathsf{SP} \iff [S] \in \mathsf{\textbf{SHP}})$.

Proof. By mutual implication.

($\Rightarrow$) Let S be an arbitrary safety property. We want to show that $[S]$ is a safety hyperproperty—that is, any trace property T not in $[S]$ contains some bad thing.

First, we find a bad thing M for T. By the definition of lifting, $[S] = \mathcal{P}(S) = \{P \in \mathsf{Prop} \mid P \subseteq S\}$. Since T is not in this set, $T \not\subseteq S$. So some trace t is in T but not in S. By the definition of safety, if $t \notin S$, there is some finite trace m that is a bad thing for S. So no extension of m is in S. Define M to be $\{m\}$.

Second, we show that M is irremediable. Note that $M \leq T$ because $m \leq t$ and $t \in T$. Let T' be an arbitrary trace property that extends M—that is, $M \leq T'$. By the definition of $\leq$, there exists a $t' \in T'$ such that $m \leq t'$. We established above that no extension of m is in S, so $t' \notin S$. But, again by the definition of lifting, $T' \notin [S]$, since T' contains a trace not in S.

Thus, by definition, $[S]$ is hypersafety.

($\Leftarrow$) Let S be an arbitrary trace property such that $[S]$ is hypersafety. We want to show that S is safety. Our strategy is as above—we find a bad thing and then show that it is irremediable.

Consider any t such that $t \notin S$. By the definition of lifting, we have that $\{t\} \notin [S]$. By the definition of hypersafety applied to $[S]$, there exists an $M \le \{t\}$ such that for all $T' \ge M$, we have $T' \notin [S]$.

We claim that M must be non-empty. To show this, suppose for sake of contradiction that M is empty. Then M is a prefix of every trace property T', so no T' can be a member of S, which implies that $[S]$ itself must be empty. But $[S] = \mathcal{P}(S)$, so $[S]$ must at least contain S as a member. This is a contradiction, thus M is non-empty and contains at least one trace.

All traces in M must be prefixes of t, by the definition of $\le$. Choose the longest such prefix in M and denote it as m^*. This m^* serves as a bad thing for t, as we show next.

Let t' be arbitrary such that $m^* \le t'$, and let $T' = \{t'\}$. By the transitivity of $\le$, we have $M \le T'$, so $T' \notin [S]$ by the above application of the definition of hypersafety. But this implies that $t' \notin S$, by the definition of lifting.

We have shown that, for any $t \notin S$, there exists an $m \le t$, such that for any $t' \ge m$, we have $t' \notin S$. Therefore, S is safety, by definition. $\square$

Theorem 4.1. $\mathsf{SHP} \subset \mathsf{SSC}$.

Proof. Assume that $\boldsymbol{S}$ is hypersafety. For sake of contradiction, also assume that $\boldsymbol{S}$ is not subset closed. This latter assumption implies that there exist two trace properties T and T' such that $T \in \boldsymbol{S}$, and $T' \notin \boldsymbol{S}$, yet $T' \subset T$. By the definition of hypersafety, since $T' \notin \boldsymbol{S}$, there exists an observation M that is a bad thing for T'—that is, $M \le T'$ and for all T'' such that $M \le T''$, it holds that $T'' \notin \boldsymbol{S}$. Consider this M. By the definition of $\le$, since $T' \subset T$ and $M \le T'$, we have $M \le T$. Then T is an instance of T'' above, which means $T \notin \boldsymbol{S}$. But this contradicts $T \in \boldsymbol{S}$. Therefore, $\boldsymbol{S}$ must be subset closed.

To see that the subset relation is strict, define the trace property $true$ as Ψ_{inf}. Consider any liveness property L other than $true$—for example, guaranteed service GS (4.1.3). When lifted to hyperproperty $[L]$, the result is subset closed by definition of $[\cdot]$. By proposition 4.2 below (whose proof does not depend on this theorem), $[L]$ is hyperliveness. Since L is not $true$, we have that $[L]$ is not ***true***, which is the only hyperproperty that is both hypersafety and hyperliveness. So $[L]$ cannot be hypersafety. Thus $[L]$ is a hyperproperty that is not hypersafety but is subset closed. $\square$

Theorem 4.2. $(\forall\, S \in \mathsf{Sys}, \boldsymbol{K} \in \mathsf{KSHP}(k) : (\exists\, K \in \mathsf{SP} : S \models \boldsymbol{K} \iff S^k \models K))$.

Proof. Let $\boldsymbol{K}$ be an arbitrary k-safety hyperproperty of system S. Our strategy is to construct a safety property K that holds of system S^k exactly when $\boldsymbol{K}$ holds of S.

Since $\boldsymbol{K}$ is k-safety, every trace property not contained in it has some bad thing of size at most k—that is, for all $T \notin \boldsymbol{K}$, there exists an observation M where $|M| \leq k$ and $M \leq T$, such that for all T' where $M \leq T'$, it holds that $T' \notin \boldsymbol{K}$. Construct the set $\boldsymbol{M}$ of all such bad things:

$$\begin{aligned}\boldsymbol{M} \;\triangleq\; \{M \in \mathsf{Obs} \mid |M| \leq k \,\wedge\, (\exists\, T \in \mathsf{Prop} : T \notin \boldsymbol{K} \,\wedge\, M \leq T)\\ \wedge\, (\forall\, T' \in \mathsf{Prop} : M \leq T' \implies T' \notin \boldsymbol{K})\}.\end{aligned}$$

Next we define some notation to encode a set of traces as a single trace. Consider a trace property T such that $|T| \leq k$. Construct a finite list of traces $t_1, t_2, \ldots, t_k$ such that $t_i \in T$ for all i. Further, we require that no t_i is equal to any t_l, for any i and l, unless $|T| < k$. We construct a trace t such that $t[j]$ is the tuple $(t_1[j], t_2[j], \ldots, t_k[j])$; note that t is a trace over state space Σ^k. Let trace t so constructed from T be denoted $zip_k(T)$, and let the inverse of this construction be denoted $\mathit{unzip}_k(t)$; note that $\mathit{zip}_k(\cdot)$ and $\mathit{unzip}_k(\cdot)$ are partial functions. We can also apply this notation to observations, which are finite sets of finite traces.[22]

Now we can construct safety property K. Let K be the set of traces over Σ^k such that no trace in K encodes an extension of any bad thing $M \in \boldsymbol{M}$:

$$K \triangleq \{t^k \mid \neg(\exists\, M \in \mathsf{Obs} : M \in \boldsymbol{M} \ \wedge\ \mathit{zip}_k(M) \leq t^k)\},$$

where t^k denotes a trace t over space Σ^k.

To see that K is safety, suppose that $t^k \notin K$. Then by the definition of K, there must exist some $M \in \boldsymbol{M}$ such that $\mathit{zip}_k(M) \leq t^k$. Consider any trace $u^k \geq \mathit{zip}_k(M)$. By the definition of K, we have that $u^k \notin K$. Thus, for any trace t^k not in K, there is some finite bad thing $\mathit{zip}_k(M)$, such that no extension u^k of the bad thing is in K. By definition, K is therefore safety.

Finally, we need to show that S satisfies $\boldsymbol{K}$ exactly when S^k satisfies K. We do so by mutual implication.

($\Rightarrow$) Suppose $S \models \boldsymbol{K}$. Then, by definition, $S \in \boldsymbol{K}$. For sake of contradiction, suppose that $S^k \not\subseteq K$. Then, by the definition of subset, there exists some $t^k \in S^k$ such that $t^k \notin K$. Let T be $\mathit{unzip}_k(t^k)$. By the definition of K, there must exist some $M \in \boldsymbol{M}$ such that $\mathit{zip}_k(M) \leq t^k$. Applying $\mathit{unzip}_k(\cdot)$ to this predicate, and noting that unzip is monotonic with respect to $\leq$, we obtain $M \leq \mathit{unzip}_k(t^k)$. By the definition of T, we then have that $M \leq T$. By the construction of $\boldsymbol{M}$, T therefore cannot be in $\boldsymbol{K}$. By the construction of S^k and the definition of T, each trace in T must also be a trace of S. So by definition, $T \leq S$. By transitivity, we have that $M \leq S$. By the

[22] In this case, the t_i have finite and potentially differing length. So if $j > |t_i|$, let $t_i[j] = \bot$ for some new state $\bot \notin \Sigma$. Thus, $\mathit{zip}_k(T)$ is a trace over state space $(\Sigma \cup \bot)^k$. We redefine trace prefix $\leq$ over this space to ignore $\bot$: let $t \leq t'$ iff, for some t'' that is a trace over Σ, $\lceil t \rceil = \lceil t' \rceil t''$, where $\lceil t \rceil$ is the truncation of t that removes any $\bot$ states. For notational simplicity, we omit this technicality in the remainder of the proof.

construction of $\boldsymbol{M}$, S then cannot be in $\boldsymbol{K}$. But this contradicts the fact that $S \in \boldsymbol{K}$. Therefore, $S^k \subseteq K$, so by definition $S^k \models K$.

($\Leftarrow$) Suppose $S^k \models K$. Then, by definition, $S^k \subseteq K$. Suppose, for sake of contradiction, that S does not satisfy $\boldsymbol{K}$. Then, by definition, $S \notin \boldsymbol{K}$. Since $\boldsymbol{K}$ is k-safety, this means that there exists an $M \leq S$, where $|M| \leq k$, such that for all $T' \geq M$, $T' \notin \boldsymbol{K}$. Let m^k be $zip_k(M)$, and let s^k be a trace of S^k such that $m^k \leq s^k$ (such a trace must exist since $M \leq S$). By the construction of K, for any $t^k \geq m^k$, we have that $t^k \notin K$. Therefore, $s^k \notin K$, and it follows that $S^k \not\subseteq K$. But this contradicts the fact that $S^k \subseteq K$. Therefore, $S \in \boldsymbol{K}$, so by definition $S \models \boldsymbol{K}$. □

Proposition 4.2. $(\forall\, L \in \mathsf{Prop} : L \in \mathsf{LP} \iff [L] \in \mathbf{LHP})$.

Proof. By mutual implication.

($\Rightarrow$) Let L be an arbitrary liveness property. We want to show that $[L]$ is a liveness hyperproperty—that is, any observation M can be extended to a trace property T that is contained in $[L]$. So let M be an arbitrary observation. By the definition of liveness, for each $m \in M$, there exists some $t \geq m$ such that $t \in L$. For a given m, let that trace t be denoted t_m. Construct the set $T = \bigcup_{m \in M}\{t_m\}$. Since all the t_m are elements of L, we have $T \subseteq L$. By the definition of lifting, it follows that T is contained in $[L]$. Further, T extends M by the construction of T. Thus, T satisfies the requirements of the trace property we needed to construct. By definition, $[L]$ is hyperliveness.

($\Leftarrow$) Let L be an arbitrary property such that $[L]$ is hyperliveness. We want to show that L is liveness. So consider an arbitrary trace t, and let $T = \{t\}$. Since $[L]$ is hyperliveness, we have that there exists a T' such that $T \leq T'$

and $T' \in [L]$. Since $T \leq T'$ and $T = \{t\}$, there exists a t' such that $t \leq t'$ and $t' \in T'$, by the definition of $\leq$. By the definition of lifting, if $t' \in T' \in [L]$, it must be the case that $t' \in L$. Thus, for any t, there exists a t' such that $t \leq t'$ and $t' \in L$. Therefore, L is liveness, by definition. $\square$

Theorem 4.3. $\mathsf{PIF} \subset \mathsf{LHP}$.

Proof. Let $\boldsymbol{P}$ be an arbitrary possibilistic information-flow hyperproperty, and let $Cl_{\boldsymbol{P}}$ be the closure operator that Mantel [78] would associate with $\boldsymbol{P}$.[23] Then, by Mantel's Definition 10, it must be the case that $\boldsymbol{P} = \{Cl_{\boldsymbol{P}}(T) \mid T \in \mathsf{Prop}\}$. Closure operators must satisfy the axiom $(\forall X : X \subseteq Cl(X))$, which we use below.

To show that $\boldsymbol{P}$ is hyperliveness, let $T \in \mathsf{Obs}$ be arbitrary. By the definition of hyperliveness, we need to show that there exists a $T' \in \mathsf{Prop}$ such that $T \leq T'$ and $T' \in \boldsymbol{P}$. Let T' be $Cl_{\boldsymbol{P}}(\hat{T})$, where $\hat{T}$ denotes the embedding of T into Prop by infinitely stuttering the final state of each trace in T, as discussed in §4.1. By the closure axiom above, we have that $\hat{T} \subseteq Cl_{\boldsymbol{P}}(\hat{T})$. So by the definition of $\leq$, we can conclude $T \leq Cl_{\boldsymbol{P}}(\hat{T}) = T'$. Further, T' must be an element of $\boldsymbol{P}$ since it is the $Cl_{\boldsymbol{P}}$-closure of trace property $\hat{T}$. Therefore, T' satisfies the required conditions, and $\boldsymbol{P}$ is hyperliveness.

To see that the subset relation is strict, consider liveness property GS (guaranteed service) from §4.1. It corresponds to liveness hyperproperty $[GS]$, but has no corresponding closure operator. For suppose that such a closure operator did exist, and consider an infinite trace t in which service fails to occur. The closure of any set containing t must still contain t, by the axiom above. But then

[23]More precisely, Mantel argues that every "possibilistic information-flow property [*sic*]" can be expressed as a *basic security predicate*, and that each basic security predicate induces a set of closure operators. Any element of this set suffices to instantiate $Cl_{\boldsymbol{P}}$. Also, Mantel's closure operators were over finite traces, and we have generalized to infinite traces.

the closure does not satisfy GS, and so the closure operator cannot correspond to $[GS]$. $\square$

Proposition 4.3. $\boldsymbol{\mathcal{O}}^B = \boldsymbol{\mathcal{O}}^{SB}$.

Proof. By mutual containment.

($\supseteq$) By definition, the elements of $\boldsymbol{\mathcal{O}}^B$ are finite intersections of elements of $\boldsymbol{\mathcal{O}}^{SB}$. Thus, every element of $\boldsymbol{\mathcal{O}}^{SB}$ is already trivially an element of $\boldsymbol{\mathcal{O}}^B$.

($\subseteq$) Let $\boldsymbol{N}$ be an arbitrary element of $\boldsymbol{\mathcal{O}}^B$. By the definition of a base, we can write $\boldsymbol{N}$ as $\bigcap_i \uparrow M_i$, where i ranges over a finite index set and each M_i is an observation. We want to show that there exists an element $\uparrow N$ of $\boldsymbol{\mathcal{O}}^{SB}$ such that $\boldsymbol{N} = \uparrow N$. So consider $\boldsymbol{N}$. Every trace property T in it must extend every M_i. Thus, by the definition of $\leq$, every such trace property T extends $\bigcup_i M_i$. Therefore $\boldsymbol{N} = \uparrow \bigcup_i M_i$. Our desired observation N is thus $\bigcup_i M_i$. Note that, for N to be a valid observation, it must be a finite set. The union over M_i must therefore result in a finite set—which it does, since i ranges over a finite index set. $\square$

Proposition 4.4. $\mathsf{SHP} = \boldsymbol{\mathcal{C}}$.

Proof. By mutual containment.

($\subseteq$) Let $\boldsymbol{S}$ be an arbitrary safety hyperproperty. We need to show that it is also a closed set. By the definition of closed, this is equivalent to showing that $\boldsymbol{S}$ is the complement of an open set. Our strategy is to construct hyperproperty $\boldsymbol{O}$, show that $\overline{\boldsymbol{O}}$ and $\boldsymbol{S}$ are equal, and show that $\boldsymbol{O}$ is open.

By the definition of hypersafety, we have that any trace property T that is not a member of $\boldsymbol{S}$—and thus is a member of $\overline{\boldsymbol{S}}$—must contain some bad thing. Consider the set $\boldsymbol{M} \in \mathcal{P}(\mathsf{Obs})$ of all bad things for $\boldsymbol{S}$. $\boldsymbol{M}$ contains one or more elements for every trace property in $\overline{\boldsymbol{S}}$:

$$\begin{aligned}\boldsymbol{M} \;\triangleq\; \{M \in \mathsf{Obs} \mid (\exists\, T \in \overline{\boldsymbol{S}} : M \leq T \\ \wedge\; (\forall\, T' \in \mathsf{Prop} : M \leq T' \implies T' \in \overline{\boldsymbol{S}}))\}.\end{aligned}$$

Next, define $\boldsymbol{O}$ as the completion of $\boldsymbol{M}$—that is, the set of all trace properties that extend a bad thing for $\boldsymbol{S}$:

$$\begin{aligned}\boldsymbol{O} &\triangleq \bigcup_{M \in \boldsymbol{M}} \uparrow M \\ &= \{T \mid (\exists\, M \in \boldsymbol{M} : M \leq T)\}, \qquad (4.A.1)\end{aligned}$$

where the equality follows by the definition of $\uparrow M$. Since each such trace property T violates $\boldsymbol{S}$, we would suspect that $\boldsymbol{O}$ is the complement of $\boldsymbol{S}$. This is indeed the case:

Claim. $\boldsymbol{O} = \overline{\boldsymbol{S}}$

Proof. (By mutual containment.)

($\subseteq$) Suppose $T \in \boldsymbol{O}$. Then by equation 4.A.1, there is some $M \in \boldsymbol{M}$ such that $M \leq T$. By the definition of $\boldsymbol{M}$, any extension of M is an element of $\overline{\boldsymbol{S}}$. Since T is such an extension, $T \in \overline{\boldsymbol{S}}$.

($\supseteq$) Suppose $T \in \overline{\boldsymbol{S}}$. Then $T \notin \boldsymbol{S}$, so by the definition of hypersafety, $(\exists\, M \in \mathsf{Obs} : M \leq T \;\wedge\; (\forall\, T' \in \mathsf{Prop} : M \leq T' \implies T' \notin \boldsymbol{S}))$. Consider that M. It must be a member of $\boldsymbol{M}$, by definition. Since $M \leq T$, we have that $T \in \boldsymbol{O}$ by equation 4.A.1. □

All that remains is to show that $\boldsymbol{O}$ is open. First, note that $\uparrow M$, for any $M \in \mathsf{Obs}$, is by definition an element of $\boldsymbol{\mathcal{O}}^{SB}$. Thus each of the sets $\uparrow M$ in the definition of $\boldsymbol{O}$ is open. Second, by the definition of open sets, a union of open sets is open. $\boldsymbol{O}$ is such a union, and is therefore open.

($\supseteq$) Let $\boldsymbol{C}$ be an arbitrary closed set. We need to show that it is also hypersafety. Our strategy is to identify, for any trace property T not in $\boldsymbol{C}$, a bad thing for T. If such a bad thing exists for all T, then $\boldsymbol{C}$ is by definition hypersafety.

Since $\boldsymbol{C}$ is closed, it is by definition the complement of an open set. By proposition 4.3, we can therefore write $\overline{\boldsymbol{C}}$ as follows:

$$\overline{\boldsymbol{C}} = \bigcup_i \uparrow M_i, \tag{4.A.2}$$

where each M_i is an observation.

Let T be an arbitrary trace property such that $T \notin \boldsymbol{C}$, or equivalently, such that $T \in \overline{\boldsymbol{C}}$. Then T must be in at least one of the infinite unions in equation 4.A.2. Thus, there must exist an i such that

$$T \in \uparrow M_i \qquad \text{and} \qquad M_i = \{U \in \mathsf{Prop} \mid M_i \leq U\}, \tag{4.A.3}$$

where the equality follows from the definition of $\uparrow$.

We construct the bad thing M for T by defining:

$$M \triangleq M_i.$$

We have that $M \leq T$, because of equation 4.A.3.

To show that M is a bad thing for T, consider any $T' \geq M$. By the definition of M, $T' \geq M_i$. By equation 4.A.3, it follows that T', like T, is a member of $\uparrow M_i$. By equation 4.A.2, $T' \in \overline{\boldsymbol{C}}$. Therefore, $T' \notin \boldsymbol{C}$.

We have now shown that for any $T \notin \boldsymbol{C}$, there exists an $M \leq T$, such that for all $T' \geq M$, $T' \notin \boldsymbol{C}$. Thus $\boldsymbol{C}$ is hypersafety, by definition. □

Proposition 4.5. $\mathsf{LHP} = \boldsymbol{\mathcal{D}}$.

Proof. By mutual containment.

($\subseteq$) Let $\boldsymbol{L}$ be an arbitrary liveness hyperproperty. We need to show that $\boldsymbol{L}$ is dense. By the definition of dense, we must therefore show that $\boldsymbol{L}$ intersects every non-empty open set. So let $\boldsymbol{O}$ be an arbitrary non-empty open set. We need to show that $\boldsymbol{L} \cap \boldsymbol{O}$ is non-empty. By proposition 4.3 and the definition of open, we can write $\boldsymbol{O}$ as $\bigcup_i \uparrow M_i$. Consider an arbitrary M_i. Since $\boldsymbol{L}$ is hyperliveness, there exists a $T \geq M_i$ such that $T \in \boldsymbol{L}$. Further, by the definition of $\uparrow$, we have that $T \in \boldsymbol{O}$. Therefore, $T \in \boldsymbol{L} \cap \boldsymbol{O}$, and it follows that $\boldsymbol{L}$ is dense, by definition.

($\supseteq$) Let $\boldsymbol{D}$ be an arbitrary dense set. To show that $\boldsymbol{D}$ is hyperliveness, we must show that any observation T can be extended to a trace property T' contained in $\boldsymbol{D}$—that is, $(\forall T \in \mathsf{Obs} : (\exists T' \in \mathsf{Prop} : T \leq T' \ \wedge \ T' \in \boldsymbol{D}))$. So let T be an arbitrary observation. Let $\boldsymbol{O}_T$ be the completion of T:

$$\begin{aligned} \boldsymbol{O}_T &\triangleq \uparrow T \\ &= \{T' \in \mathsf{Prop} \mid T \leq T'\} \end{aligned} \tag{4.A.4}$$

$\boldsymbol{O}_T$ is an element of $\boldsymbol{\mathcal{O}}^{SB}$, the subbase of our topology, by definition. Thus, by the definition of a subbase, $\boldsymbol{O}_T$ is an open set. By the definition of a dense set (which is that a dense set intersects every open set), we therefore have that $\boldsymbol{O}_T \cap \boldsymbol{D} \neq \emptyset$. Let T' be any element in the set $\boldsymbol{O}_T \cap \boldsymbol{D}$. By equation 4.A.4, we have $T \leq T'$.

We have now shown that, for an arbitrary observation T, there exists a trace property T' such that $T \leq T'$ and $T' \in \boldsymbol{D}$. Therefore, $\boldsymbol{D}$ is hyperliveness, by definition. $\square$

Theorem 4.4. $\boldsymbol{\mathcal{O}} = \mathfrak{V}_L(\mathcal{O})$.

Proof. By mutual containment.

($\subseteq$) Suppose $\boldsymbol{O} \in \boldsymbol{\mathcal{O}}$. By the definitions of a base and of $\boldsymbol{\mathcal{O}}$, we can write $\boldsymbol{O}$ as $\bigcup_i^\infty \uparrow M_i$, where each M_i is an element of Obs.[24] Now we calculate:

$$\bigcup_i^\infty \uparrow M_i$$

$= \quad \langle$ definition of $\uparrow \rangle$

$$\bigcup_i^\infty \{T \mid T \geq M_i\}$$

$= \quad \langle$ definition of $\leq \rangle$

$$\bigcup_i^\infty \{T \mid (\forall^* \, m_{ij} \in M_i \, : \, (\exists \, t \in T \, : \, m_{ij} \leq t))\}$$

$= \quad \langle$ definition of $\uparrow \rangle$

$$\bigcup_i^\infty \{T \mid (\forall^* \, m_{ij} \in M_i \, : \uparrow m_{ij} \cap T \neq \emptyset)\}$$

$= \quad \langle$ definition of $\langle \cdot \rangle$ $\rangle$

$$\bigcup_i^\infty \{T \mid (\forall^* \, m_{ij} \in M_i \, : \, T \in \langle \uparrow m_{ij} \rangle)\}$$

$= \quad \langle$ definition of $\cap$ $\rangle$

$$\bigcup_i^\infty \bigcap_j^* \langle \uparrow m_{ij} \rangle$$

Since $\uparrow m_{ij} \in \mathcal{O}^B$ by definition, and $\mathcal{O}^B \subseteq \mathcal{O}$ by the definition of base, we have that $\langle \uparrow m_{ij} \rangle \in \mathfrak{V}_L^{SB}(\mathcal{O})$. Thus, by the definition of subbase, $\bigcup_i^\infty \bigcap_j^* \langle \uparrow m_{ij} \rangle \in \mathfrak{V}_L(\mathcal{O})$. Therefore, by the calculation above, we can conclude $\boldsymbol{O} \in \mathfrak{V}_L(\mathcal{O})$.

[24]We decorate quantifiers with ∞ and $*$ to denote an infinite and finite range, respectively.

($\supseteq$) Suppose $\boldsymbol{O} \in \mathfrak{V}_L(\mathcal{O})$. By the definition of subbase and $\mathfrak{V}_L$, we can write $\boldsymbol{O}$ as $\bigcup_i^\infty \bigcap_j^* \langle O_{ij} \rangle$, where each O_{ij} is an element of $\mathcal{O}$. Now we calculate:

$\bigcup_i^\infty \bigcap_j^* \langle O_{ij} \rangle$

$=$ $\langle$ definition of $\langle \cdot \rangle$ $\rangle$

$\bigcup_i^\infty \bigcap_j^* \{T \mid T \cap O_{ij} \neq \emptyset\}$

Since O_{ij} is open in the base topology $\mathcal{O}$, it can be rewritten a union of base open sets $\uparrow t_{ijk}$, where each t_{ijk} is a finite trace:

$$O_{ij} \;=\; \bigcup_k^\infty \uparrow t_{ijk}.$$

We continue calculating:

$=$ $\langle$ rewriting O_{ij} $\rangle$

$\bigcup_i^\infty \bigcap_j^* \{T \mid T \cap (\bigcup_k^\infty \uparrow t_{ijk}) \neq \emptyset\}$

$=$ $\langle$ set theory $\rangle$

$\bigcup_i^\infty \{T \mid (\forall^* j \,:\, (\exists^\infty k \,:\, T \cap \uparrow t_{ijk} \neq \emptyset))\}$

$=$ $\langle$ definition of $\leq$ $\rangle$

$\bigcup_i^\infty \{T \mid (\forall^* j \,:\, (\exists^\infty k \,:\, \{t_{ijk}\} \leq T))\}$

$=$ $\langle$ set theory; let k' be the k guaranteed to exist for i and j $\rangle$

$\bigcup_i^\infty \{T \mid \bigcup_j^* t_{ijk'} \leq T\}$

$=$ $\langle$ let $M_i = \bigcup_j^* t_{ijk'}$; definition of $\uparrow$ $\rangle$

$\bigcup_i^\infty \uparrow M_i$

Finally, since M_i is a finite set of finite traces, it is an element of Obs. So by definition, $\uparrow M_i \in \boldsymbol{\mathcal{O}}^{SB}$. Thus by the definition of base, $\bigcup_i^\infty \uparrow M_i \in \boldsymbol{\mathcal{O}}$. Therefore, by the calculation above, we can conclude $\boldsymbol{O} \in \boldsymbol{\mathcal{O}}$. $\square$

Proposition 4.6. $\mathsf{SHP} = Cl_C(\{[S] \mid S \in \mathsf{SP}\})$.

Proof. Let $\boldsymbol{S}$ be an arbitrary safety hyperproperty. By proposition 4.4, $\boldsymbol{S}$ is a closed set in topology $\boldsymbol{\mathcal{O}}$. By theorem 4.4, $\boldsymbol{S}$ is thus also a closed set in topology $\mathfrak{V}_L(\mathcal{O})$. By the definition of closed, $\boldsymbol{S}$ is the complement of an open set in topology $\mathfrak{V}_L(\mathcal{O})$. By the definition of a base, we can thus write $\overline{\boldsymbol{S}}$ as unions of intersections of base elements. Letting $\sim$ denote set complement, we calculate:

$\overline{\boldsymbol{S}}$

$=$ $\langle$ definition of base $\rangle$

$\bigcup_i^\infty \bigcap_j^* \langle O_{ij} \rangle$

$=$ $\langle$ definition of $\langle\cdot\rangle$ $\rangle$

$\bigcup_i^\infty \bigcap_j^* \{T \mid T \cap O_{ij} \neq \emptyset\}$

$=$ $\langle$ double negation $\rangle$

$\sim\sim\bigcup_i^\infty \bigcap_j^* \{T \mid T \cap O_{ij} \neq \emptyset\}$

$=$ $\langle$ set theory $\rangle$

$\sim\bigcap_i^\infty \bigcup_j^* \{T \mid T \cap O_{ij} = \emptyset\}$

$=$ $\langle$ set theory $\rangle$

$\sim\bigcap_i^\infty \bigcup_j^* \{T \mid T \subseteq \overline{O_{ij}}\}$

$=$ $\langle$ definition of $[\cdot]$ $\rangle$

$\sim\bigcap_i^\infty \bigcup_j^* [\overline{O_{ij}}]$

Removing a complement from each side of the above equation, we obtain

$$\boldsymbol{S} = \bigcap_i^\infty \bigcup_j^* [\overline{O_{ij}}].$$

Since each O_{ij} is open in topology $\mathcal{O}$, we have that $\overline{O_{ij}}$ is closed in $\mathcal{O}$. By the fact that closed sets in $\mathcal{O}$ correspond to safety properties [5], $\overline{O_{ij}}$ is a safety property. Therefore, $\boldsymbol{S}$ is the infinite intersection of finite unions of safety properties, and by definition of Cl_C must be an element of $Cl_C(\{[S] \mid S \in \mathsf{SP}\})$.

Similarly, given an arbitrary element of $Cl_C(\{[S] \mid S \in \mathsf{SP}\})$, the same reasoning used above establishes that it is also an element of **SHP**. Therefore, by mutual containment, the two sets are equal. □

Theorem 4.5. $(\forall\, \boldsymbol{P} \in \mathsf{HP} : (\exists\, \boldsymbol{S} \in \mathsf{SHP}, \boldsymbol{L} \in \mathsf{LHP} : \boldsymbol{P} = \boldsymbol{S} \cap \boldsymbol{L}))$.

Proof. This theorem can be easily proved by adapting either the logical [105] or topological [5] proof of the intersection theorem for trace properties. The domains involved are merely upgraded to include an additional level of sets. Here we take the former approach and rehearse the logical proof.

Our strategy is as follows. Given hyperproperty $\boldsymbol{P}$, we construct safety hyperproperty $\boldsymbol{S}$ that contains $\boldsymbol{P}$ as a subset. We also construct liveness hyperproperty $\boldsymbol{L}$ that contains $\boldsymbol{P}$. The intersection of $\boldsymbol{S}$ and $\boldsymbol{L}$ then necessarily contains $\boldsymbol{P}$, and we shall show that the intersection is, in fact, exactly $\boldsymbol{P}$.

To construct $\boldsymbol{S}$, we define the safety hyperproperty $\mathit{Safe}(\boldsymbol{P})$, which stipulates that the hyperliveness of $\boldsymbol{P}$ is never violated. A bad thing for this safety hyperproperty is any set of traces that cannot be extended to satisfy $\boldsymbol{P}$. So we require that $\mathit{Safe}(\boldsymbol{P})$ contains only sets T of traces such that any observation of T can be extended to satisfy $\boldsymbol{P}$. Formally,

$$\begin{aligned}\mathit{Safe}(\boldsymbol{P}) \;\triangleq\; \{T \in \mathsf{Prop} \mid (\forall\, M \in \mathsf{Obs} : M \leq T \\ \implies (\exists\, T' \in \mathsf{Prop} : M \leq T' \;\wedge\; T' \in \boldsymbol{P}))\}.\end{aligned}$$

It is straightforward to establish that $\mathit{Safe}(\boldsymbol{P})$ is hypersafety: Any set T not contained in $\mathit{Safe}(\boldsymbol{P})$ must satisfy the negation of the predicate in the above definition of $\mathit{Safe}(\boldsymbol{P})$—that is, $(\exists\, M \in \mathsf{Obs} : M \le T \ \wedge\ (\forall\, T' \in \mathsf{Prop} : M \le T' \implies T' \notin \boldsymbol{P}))$. If no extension of M can be in $\boldsymbol{P}$, then no extension T' of M can be in $\mathit{Safe}(\boldsymbol{P})$ because the hyperliveness of $\boldsymbol{P}$ would be violated in T' at observation M. So

$$\begin{aligned}&(\forall\, T' \in \mathsf{Prop} : M \le T' \implies T' \notin \boldsymbol{P})\\ &\qquad\qquad \implies (\forall\, T' \in \mathsf{Prop} : M \le T' \implies T' \notin \mathit{Safe}(\boldsymbol{P})). \quad (4.A.5)\end{aligned}$$

Thus, by monotonicity, $(\exists\, M \in \mathsf{Obs} : M \le T \ \wedge\ (\forall\, T' \in \mathsf{Prop} : M \le T' \implies T' \notin \mathit{Safe}(\boldsymbol{P})))$. Therefore $\mathit{Safe}(\boldsymbol{P})$ is hypersafety.

Similarly, to construct $\boldsymbol{L}$, we define the liveness hyperproperty $\mathit{Live}(\boldsymbol{P})$, which stipulates that it is always possible either to satisfy $\boldsymbol{P}$ or to become impossible, due to some bad thing, to satisfy $\boldsymbol{P}$. In the latter case, a safety hyperproperty has been violated—namely, $\mathit{Safe}(\boldsymbol{P})$. Formally,

$$\mathit{Live}(\boldsymbol{P}) \ \triangleq\ \boldsymbol{P} \cup \overline{\mathit{Safe}(\boldsymbol{P})},$$

where $\overline{\boldsymbol{H}}$ denotes the complement of hyperproperty $\boldsymbol{H}$ with respect to Prop. To show that $\mathit{Live}(\boldsymbol{P})$ is hyperliveness, consider any observation T. Suppose that T can be extended to some trace property T' such that $T' \in \boldsymbol{P}$. Then T' is also in $\mathit{Live}(\boldsymbol{P})$, so $\mathit{Live}(\boldsymbol{P})$ is hyperliveness for T. On the other hand, if T cannot be extended to satisfy $\boldsymbol{P}$, then T is a bad thing for $\mathit{Safe}(\boldsymbol{P})$—that is, $(\forall\, T' \in \mathsf{Prop} : T \le T' \implies T' \notin \boldsymbol{P})$. Let T' be an arbitrary extension of T. By the same reasoning as equation (4.A.5), T' is not in $\mathit{Safe}(\boldsymbol{P})$. Therefore T' must be in $\overline{\mathit{Safe}(\boldsymbol{P})}$. Thus, $\mathit{Live}(\boldsymbol{P})$ is again hyperliveness for T. We conclude that $\mathit{Live}(\boldsymbol{P})$ is hyperliveness.

Next, note that $\boldsymbol{P} \subseteq \mathit{Safe}(\boldsymbol{P})$, because any element T of $\boldsymbol{P}$ satisfies the definition of $\mathit{Safe}(\boldsymbol{P})$. In particular, for any $M \leq T$, there is a $T' \geq M$ such that $T' \in \boldsymbol{P}$—namely, $T' = T$. Thus, $\mathit{Safe}(\boldsymbol{P}) = \boldsymbol{P} \cup \mathit{Safe}(\boldsymbol{P})$.

Finally, let $\boldsymbol{S} = \mathit{Safe}(\boldsymbol{P})$ and $\boldsymbol{L} = \mathit{Live}(\boldsymbol{P})$, and we prove the theorem by simple set manipulation:

$$\begin{aligned}
\boldsymbol{S} \cap \boldsymbol{L} &= \mathit{Safe}(\boldsymbol{P}) \cap \mathit{Live}(\boldsymbol{P}) \\
&= (\boldsymbol{P} \cup \mathit{Safe}(\boldsymbol{P})) \cap (\boldsymbol{P} \cup \overline{\mathit{Safe}(\boldsymbol{P})}) \\
&= \boldsymbol{P} \cap (\mathit{Safe}(\boldsymbol{P}) \cup \overline{\mathit{Safe}(\boldsymbol{P})}) \\
&= \boldsymbol{P} \cap \mathsf{Prop} \\
&= \boldsymbol{P}
\end{aligned}$$

□

CHAPTER 5

FORMALIZATION OF SYSTEM REPRESENTATIONS

Security policies are *properties* of systems, meaning that a system either does or does not satisfy a security policy. Chapter 4 models systems (and their executions) with trace sets. Some models of system execution are expressed with other mathematical formalisms—for example, relational semantics, labeled transition systems, and state machines. And probability can be used with each of these formalisms to model random behaviors of systems. Chapter 4 mentions some of these formalisms but does not make them precise.

For example, recall *noninterference* stipulates that commands executed on behalf of users holding high clearances have no effect on system behavior observed by users holding low clearances. Goguen and Meseguer's definition of noninterference [46] models system behavior with state machines, whereas our definition ***GMNI*** (4.1.4), repeated below, assumes an encoding of state machines as trace sets and requires a trace set T to contain, for any trace t, a corresponding trace t' with no high input events yet with the same low input and output events as t:

$$\begin{aligned}\boldsymbol{GMNI} \;\triangleq\; \{T \in \mathsf{Prop} \mid T \in \boldsymbol{SM} \\ \wedge\ (\forall\, t \in T : (\exists\, t' \in T : ev_{Hin}(t') = \epsilon \\ \wedge\ ev_L(t) = ev_L(t')))\}.\end{aligned}$$

Conjunct $T \in \boldsymbol{SM}$ expresses the requirement that trace set T encodes a state machine, but we have not yet defined set ***SM*** (we shall in §5.4). Nor have we classified ***GMNI*** as hypersafety or hyperliveness.

It is reasonable to expect that ***GMNI*** is hypersafety; the bad thing should be a set $\{t, t'\}$ of finite traces where t' contains no high inputs and contains the same low inputs as t, yet t and t' have different low outputs. But ***GMNI*** fails to be hypersafety because of a technicality. Goguen and Meseguer's state machines must be deterministic, so ***SM*** must exclude all trace sets that exhibit nondeterminism. Thus a system T might fail to satisfy ***GMNI*** only because T is nondeterministic, in which case a deterministic, non-interfering observation of T would be remediable—hence ***GMNI*** would not be hypersafety.[1] The problem is that the definition of hypersafety, by quantifying over Prop, assumed that systems are allowed to be nondeterministic. Now that we are interested in state machines, our definitions of hypersafety and hyperliveness should quantify over only those trace sets that encode state machines. And in general, those definitions should be parameterized on a system representation.

This chapter proceeds as follows. The definitions of hypersafety and hyperliveness are generalized in §5.1 to account for system representations. Hyperproperties for relational systems, labeled transition systems, state machines, and probabilistic systems are presented in §5.2, §5.3, §5.4, and §5.5. The technical results of chapter 4 are generalized in §5.6 to account for system representations, and §5.7 concludes.

5.1 Generalized Hypersafety and Hyperliveness

Chapter 4 assumed a particular system representation—namely, Prop, the set of all trace sets. Now, let Rep be a set of trace sets that encodes a system representation. For example, each set in Rep might encode a state machine. Note that Rep is a subset of Prop.

[1]A similar problem would occur even if we used implication instead of conjunction in the definition of ***GMNI*** to formalize the requirement that systems be (deterministic) state machines: any observation could be remediated by adding traces that represent nondeterministic transitions of the state machine.

Recall that Obs is the set of *observations* of Prop, and that an observation is a finite set of finite traces. We now need to define the set of observations of Rep. Let $Obs(\mathsf{Rep})$ denote the subset of Obs containing observations of Rep, where

$$Obs(\mathsf{Rep}) \triangleq \{M \in \mathsf{Obs} \mid (\exists\, T \in \mathsf{Rep} : M \leq T)\}.$$

Note that $Obs(\mathsf{Rep})$ is simply Obs if Rep equals Prop.

Now we can define hypersafety and hyperliveness for a given system representation.

Definition 5.1. A hyperproperty $\boldsymbol{S}$ is a *safety hyperproperty for system representation* Rep (is *hypersafety for* Rep) iff

$$\begin{aligned}(\forall\, T \in \mathsf{Rep} : T \notin \boldsymbol{S} \implies (\exists\, M \in Obs(\mathsf{Rep}) : M \leq T \\ \wedge\ (\forall\, T' \in \mathsf{Rep} : M \leq T' \implies T' \notin \boldsymbol{S}))).\end{aligned}$$

Definition 5.2. Hyperproperty $\boldsymbol{L}$ is a *liveness hyperproperty for system representation* Rep (is *hyperliveness for* Rep) iff

$$(\forall\, T \in Obs(\mathsf{Rep}) : (\exists\, T' \in \mathsf{Rep} : T \leq T' \ \wedge\ T' \in \boldsymbol{L})).$$

Note that both definitions simplify to the original definitions of hypersafety and hyperliveness in chapter 4 if Rep equals Prop. We now demonstrate the use of these generalized definitions with several system representations.

5.2 Relational Systems

In language-based information-flow security [104], a program P is sometimes modeled (e.g., with large-step operational semantics) as a relation $\Downarrow$ such that

$\langle P, s\rangle \Downarrow s'$ if P begun in *initial* state s terminates in *final* state s'. Using this relation, noninterference can be stated as

$$s_1 =_L s_2 \wedge \langle P, s_1\rangle \Downarrow s_1' \wedge \langle P, s_2\rangle \Downarrow s_2' \implies s_1' =_L s_2',$$

where relation $=_L$ (c.f. observational determinism ***OD*** (4.1.6)) determines which states are low-equivalent. This statement of noninterference is *termination insensitive* because it allows information to leak through termination channels.

To model a program P as set T of traces, intuitively, imagine that an observer of the program periodically checks to see in what state the program is. If P begun in initial state s never terminates, the observer will see an infinite sequence containing only s. If P does terminate in final state s', the observer will see a finite sequence of s followed by an infinite sequence of s'. Let T be the set of all such traces. Formally, T is defined as follows:

$$\begin{aligned} T \;=\; & \{t \in \Psi_{\mathsf{inf}} \mid \langle P, s\rangle \Downarrow s' \wedge t \in s^+(s')^\omega\} \\ & \qquad \cup \{t \in \Psi_{\mathsf{inf}} \mid \neg(\exists\, s' : \langle P, s\rangle \Downarrow s') \wedge t = s^\omega\}. \end{aligned}$$

Let ***Rel***, the set of all *relational systems*, be the set of all trace sets so constructed for any P.

Define *termination-insensitive relational noninterference* as a hyperproperty:

$$\begin{aligned} \textbf{\textit{TIRNI}} \;\triangleq\; & \{T \in \mathsf{Prop} \mid T \in \textbf{\textit{Rel}} \\ & \wedge (\forall\, t_1, t_2 \in T : t_1[0] =_L t_2[0] \\ & \implies \mathit{diverges}(t_1) \vee \mathit{diverges}(t_2) \\ & \vee (\exists\, s_1, s_2 \in \Sigma : \mathit{terminates}(t_1, s_1) \\ & \wedge \mathit{terminates}(t_2, s_2) \wedge s_1 =_L s_2))\}. \qquad (5.2.1) \end{aligned}$$

Predicate $\mathit{diverges}(t)$ holds whenever t is a trace of a program P such that P does not terminate when begun in initial state $t[0]$, so $t = (t[0])^\omega$. Similarly, predicate

$\mathit{terminates}(t, s)$ holds whenever P terminates in final state s when begun in initial state $t[0]$, so $t = (t[0])^+ s^\omega$. We assume without loss of generality that final states are distinguishable from initial states (e.g., by having a special flag set), so that $\mathit{diverges}$ and $\mathit{terminates}$ can distinguish between nontermination and termination in a final state that otherwise is identical to an initial state. ***TIRNI*** is hypersafety for ***Rel***: the bad thing is a pair of traces that begin in low-equivalent initial states but terminate in final states that are not low-equivalent.

Termination-sensitive noninterference is the same as termination insensitive, except that it forbids one trace to diverge and the other to terminate. So define *termination-sensitive relational noninterference* as follows:

$$\begin{aligned}
\boldsymbol{\mathit{TSRNI}} \;\triangleq\; \{T \in \mathsf{Prop} \mid T \in \boldsymbol{\mathit{Rel}} \\
&\wedge\ (\forall\, t_1, t_2 \in T : t_1[0] =_L t_2[0] \\
&\quad \Longrightarrow\ (\mathit{diverges}(t_1) \wedge \mathit{diverges}(t_2)) \\
&\qquad \vee\ (\exists\, s_1, s_2 \in \Sigma : \mathit{terminates}(t_1, s_1) \\
&\qquad\quad \wedge\ \mathit{terminates}(t_2, s_2) \wedge s_1 =_L s_2))\}. \quad (5.2.2)
\end{aligned}$$

Note that the only change is that a disjunction became a conjunction. ***TSRNI*** is neither hypersafety nor hyperliveness for ***Rel***. To see that it is not hypersafety for ***Rel***, consider a system containing a pair $\{t, t'\}$ of traces, where t diverges and t' does not, yet where t and t' contain low-equivalent initial states, does not satisfy ***TSRNI***. But any finite prefix of this pair could be remediated by extending the prefix of t to terminate in the same final state as t'. Likewise, to see that ***TSRNI*** is not hyperliveness for ***Rel***,[2] consider a finite observation

[2]Terauchi and Aiken [115] characterized termination-sensitive noninterference as "2-liveness," where they defined "2-liveness" as a "property which may observe up to two possibly infinite traces to refute the property." Although they are correct that ***TSRNI*** could be refuted by observing two infinite traces, refutation is really about safety, not liveness—there is no good thing for ***TSRNI***, but there is an infinitely-observable bad thing. So "2-infinite-safety" would be a better term than "2-liveness."

containing a pair of terminating traces that have low-equivalent initial states but not low-equivalent final states. This observation cannot be extended to be in ***TSRNI***.

5.3 Labeled Transition Systems

Definitions of noninterference are sometimes based on *bisimulation*, which is a relation that specifies whether two systems are equivalent to an observer. Bisimulations are often expressed over *labeled transition systems*, which are triples $(S, L, \rightarrow)$ where S is a set of LTS-states,[3] L is a set of labels, and $\rightarrow$ is a relation on $S \times L \times S$ [90]. Elements of relation $\rightarrow$ are usually notated $s_1 \xrightarrow{\ell} s_2$ and are interpreted to mean that the system has a transition labeled ℓ from LTS-state s_1 to LTS-state s_2.

A labeled transition system $(S, L, \rightarrow)$ can be encoded as a set of traces. Define the state space Σ for the traces to be $S \times L$.[4] Given state $s \in \Sigma$, let $st(s)$ denote the LTS-state from s, and let $lab(s)$ denote the label from s. Define $traces(S, L, \rightarrow)$ to be

$$\{t \mid (\forall\, i \in \mathbb{N} : st(t[i]) \overset{lab(t[i])}{\rightarrow} st(t[i+1]))\}.^{5}$$

Let ***LTS*** be the set of all trace sets so constructed for any LTS.

Bismulation nondeducibility on compositions. We now demonstrate how to use this encoding by formalizing Focardi and Gorrieri's [44] definition of *bisimulation nondeducibility on compositions* (BNDC), which is a noninterference pol-

[3]We use the term *LTS-state* to distinguish these from the states defined in §4.1.

[4]This construction would not work with an impoverished notion of state, as observed by Focardi and Gorrieri [44] for states that are elements only of L.

[5]We could replace $lab(t[i])$ with $lab(t[i+1])$ in this definition; the choice of where to store the label is arbitrary.

icy for nondeterministic LTSs. The intuition behind this policy is that a system should appear the same to a low observer no matter with what other system it is composed (i.e., run in parallel). Assume that set L of labels can be partitioned into three sets of *actions* (i.e., events): a set of low security actions, a set H of high security actions, and $\{\tau\}$, where τ is an unobservable *internal* action. An LTS $E = (S, L, \rightarrow)$ satisfies BNDC, denoted $BNDC(E)$, iff for all LTSs $F = (S, H \cup \{\tau\}, \rightarrow_F)$ that take only high and internal actions,

$$E/H \approx (E|F) \setminus H,$$

with notations $/$, $|$, $\setminus$, and $\approx$ informally defined as follows:[6]

- Hiding operator E/H relabels as τ all actions from H that occur during execution of E. System E/H thus represents the view of system E by a low observer, since all the high actions are hidden.
- Parallel composition operator $E|F$ denotes the interleaving of systems E and F. The systems can synchronize on actions, causing the composed system to emit internal action τ.
- Restriction operator $E \setminus H$ prohibits the occurrence of any actions from H during execution of E, meaning that no transition with a label from H is allowed. System $(E|F) \setminus H$ thus represents a low observer's view of E when all the high actions that E takes are synchronized with F.
- Weak bisimulation relation $E \approx F$ intuitively means that E and F can simulate each other: if E can take a transition with label ℓ, then there must exist a transition of F that is also labeled ℓ, and after taking those transitions E and F must remain bisimilar. F is allowed to take any number

[6]The formal definitions (over LTSs) are standard and given by Focardi and Gorrieri [44]. It is straightforward to define them directly over trace sets.

of internal transitions (labeled τ) before or after the ℓ-labeled transition. Further, the relation must be symmetric, such that if $E \approx F$ then $F \approx E$.

Thus, if $E/H \approx (E|F)\backslash H$, a low observer's view of E does not change when E is composed with any high security system F. The hyperproperty corresponding to Focardi and Gorrieri's BNDC is

$$\begin{aligned}\textbf{\textit{BNDC}} \;\triangleq\; \{T \in \mathsf{Prop} \mid T \in \textbf{\textit{LTS}} \\ \wedge\ (\exists\, E \in \textbf{\textit{LTS}} : T = \mathit{traces}(E) \\ \wedge\ \mathit{BNDC}(E))\}.\end{aligned} \quad (5.3.1)$$

BNDC is hyperliveness for ***LTS*** because of the existential in definition of $\approx$: any observation can be remedied by adding additional transitions. This remediation corresponds to a closure operator because it only adds traces, thus ***BNDC*** is a possibilistic-information flow policy.

Boudol and Castellani's noninterference. Boudol and Castellani [18] define a bisimulation-based noninterference policy for concurrent programs. To model this policy as a hyperproperty, we first formalize their model of program execution. They model execution as a binary relation $\rightarrow$ on program terms and memories; a program term P and a memory μ step to a new program term P' and memory μ'. Define the set Σ_P of states for program P to be the set of pairs of a program term and a memory, $\mathit{prog}(s)$ to be the program term from state s, and $\mathit{mem}(s)$ to be the memory from state s. Define $\mathit{traces}(P)$ to be the set of all traces t such that $\mathit{prog}(t[0])$ is P, and for all i, $t[i] \rightarrow t[i+1]$. This construction encodes P as a set of traces and is an instance of our general construction for encoding LTSs (c.f. §5.3); here there are only LTS-states and no labels.

Second, we formalize Boudol and Castellani's security policy. Let $=_L$ be an equivalence relation on memories such that $\mu_1 =_L \mu_2$ means μ_1 and μ_2 are indistinguishable to a low observer. State s can *step* to state s' in program P, denoted $steps_P(s, s')$, if

$$(\exists\, t \in \Psi_{\mathsf{inf}}, i \in \mathbb{N} : t \in traces(P) \;\wedge\; t[i] = s \;\wedge\; t[i+1] = s').$$

Define $\approx_L^P$ (read "bisimilar") to be a binary relation on Σ_P such that if s_1 is bisimilar to s_2, then s_1 and s_2 must have indistinguishable memories to a low observer; further, if s_1 can step to state s_1', then either s_1' is bisimilar to s_2, or s_2 can step to s_2' where s_1' and s_2' are bisimilar. Formally, $\approx_L^P$ is the largest symmetric binary relation on Σ_P such that

$$\begin{aligned} s_1 \approx_L^P s_2 \implies\; & mem(s_1) =_L mem(s_2) \\ & \wedge (\exists\, s_1' \in \Sigma : steps_P(s_1, s_1') \implies s_1' \approx_L^P s_2 \\ & \qquad \vee\; (\exists\, s_2' \in \Sigma : steps_P(s_2, s_2') \;\wedge\; s_1' \approx_L^P s_2')). \end{aligned}$$

Relation $\approx_L^P$ formalizes Definition 3.5 ($(\Gamma, \mathcal{L})$-Bisimulation) from [18].

Boudol and Castellani define program P to be secure, which we denote $BCNI(P)$, iff P is bisimilar to itself in all initially low-equivalent memories:

$$BCNI(P) \;\triangleq\; (\forall\, \mu_1, \mu_2 : \mu_1 =_L \mu_2 \implies (P, \mu_1) \approx_L^P (P, \mu_2)).$$

$BCNI(P)$ formalizes Definition 3.8 (Secure Programs) from [18]. The hyperproperty containing all secure programs according to Boudol and Castellani's definition is

$$\boldsymbol{BCNI} \;\triangleq\; \{T \in \mathsf{Prop} \mid T \in \boldsymbol{LTS} \implies (\exists\, P : T = traces(P) \;\wedge\; BCNI(P))\}.$$

BCNI is hyperliveness because of the existential quantifier on s_2' in the definition of $\approx_L^P$: any observation that contains traces leading to non-bisimilar states can be remedied by adding additional traces leading to bisimilar states. This remediation corresponds to a closure operator because it only adds traces, thus ***BCNI*** is a possibilistic information-flow policy.

5.4 State Machines

Goguen and Meseguer [46] define a *state machine* as a tuple (S, C, O, out, do, s_0), where S is a set of machine states, C is a set of commands, O is a set of outputs, out is a function from S to O yielding what output the user of the machine observes when the machine is in a given state, do is a function from $S \times C$ to S describing how the machine transitions between states as a function of commands, and s_0 is the initial state of the machine.[7] Such state machines are deterministic because do is a function rather than a relation.

A state machine $M = (S, C, O, out, do, s_0)$ can be encoded as a set of traces. The construction proceeds in two steps. First, M is encoded as a labeled transition system (c.f. §5.3) by treating the machine commands and outputs as labels: Let the set $\hat{S}$ of LTS-states be set S of machine states. Let the set $\hat{L}$ of labels be product set $C \times O$ of commands and outputs. Let the transition relation $\rightarrow$ include $(s, (c, o), s')$ whenever $do(s, c) = s'$ and $out(s') = o$. We now have a labeled transition system $L = (\hat{S}, \hat{L}, \rightarrow)$. Second, the traces of M are the traces of L that start with s_0: let $traces(M)$ be $traces(\hat{S}, L, \rightarrow) \cap \{t \in \Psi_{\mathsf{inf}} \mid t[0] = s_0\}$.

The set ***SM*** of all state machines is a hyperproperty:

$$\boldsymbol{SM} \quad \triangleq \quad \{T \in \mathsf{Prop} \mid (\exists\, M : T = traces(M))\}. \tag{5.4.1}$$

[7]Our definition of state machines simplifies Goguen and Meseguer's by omitting user clearances, though the clearances still appear in the definition of ***GMNI***.

Finally, we can declare that ***GMNI*** is hypersafety for ***SM***, fulfilling our expectation from the beginning of this chapter.

5.5 Probabilistic Systems

A *probabilistic system* is equipped with a function p such that the system transitions from a state s to state s' with probability $p(s, s')$.[8] This probability is *Markovian* because it does not depend upon past or future states in an execution; nonetheless, dependence upon the past or future can be modeled by allowing states to contain history or prophecy variables [1]. Function p can itself even be encoded into the state in various ways. For example, state s could record $p(s, s')$ for all states s'. Or in a trace t, state $t[i]$ could record $p(t[i], t[i+1])$. This latter encoding is an instantiation of the construction in §5.3 for encoding labeled transition systems as sets of traces; here, the labels are probabilities. Either way, probabilistic systems can be modeled as sets of traces. Define ***PR*** to be the set of all trace sets that encode probabilistic systems—that is, trace set T is in ***PR*** if T encodes a valid probability function $p(\cdot, \cdot)$.

To obtain a probability measure on sets of traces, let $\mathsf{Pr}_{s,S}(T)$ denote the probability with which set T of finite traces is produced by probabilistic system S beginning in initial state s.[9] O'Neill et al. [96] show how to construct this probability measure from p. We now demonstrate how the measure can be used in the definitions of hyperproperties.

[8]To be a valid probability, $p(s, s')$ must be in the real interval [0,1] for all s and s'; and for all s, it most hold that $\sum_{s'} p(s, s') = 1$.

[9]The initial state can be eliminated if we also assume a prior probability on initial states [52, §6.5]. The requirement that the traces in T be finite is, however, essential to ensure that $\mathsf{Pr}_{s,S}(T)$ is a valid probability measure.

Probabilistic noninterference. In information-flow security, the original motivation for adding probability to system models was to address covert channels and to establish connections between information theory and information flow [48, 49, 88]. *Probabilistic noninterference* [49] emerged from this line of research. Intuitively, this policy requires that the probability of every low trace be the same for every low-equivalent initial state. To formulate probabilistic noninterference as a hyperproperty, we need some notation. Let the *low equivalence class* of a finite trace t be denoted $[t]_L$, where

$$[t]_L \quad \triangleq \quad \{t' \in \Psi_{\mathsf{fin}} \mid ev_L(t) = ev_L(t')\}.$$

The probability that system S, starting in state s, produces a trace that is low-equivalent to t is therefore $\mathsf{Pr}_{s,S}([t]_L)$. Let the set of initial states of trace property T be denoted $\mathit{Init}(T)$, where

$$\mathit{Init}(T) \quad \triangleq \quad \{s \mid \{s\} \leq T\}.$$

Probabilistic noninterference can now be expressed as follows:

$$\begin{aligned} \boldsymbol{PNI} \quad \triangleq \quad \{T \in \mathsf{Prop} \mid T \in \boldsymbol{PR} \\ \wedge\ (\forall\, s_1, s_2 \in \mathit{Init}(T) : ev_L(s_1) = ev_L(s_2) \\ \implies (\forall\, t \in \Psi_{\mathsf{fin}} : \mathsf{Pr}_{s_1,T}([t]_L) = \mathsf{Pr}_{s_2,T}([t]_L)))\}. \end{aligned} \qquad (5.5.1)$$

PNI is not hyperliveness for ***PR***, because a system that deterministically produces two non-low-equivalent traces from two initial low-equivalent states cannot be extended to satisfy ***PNI***. Whether ***PNI*** is hypersafety for ***PR*** depends on whether state space Σ is finite. To see why, consider a system T such that $T \notin$ ***PNI*** and $T \in$ ***PR***. We can attempt to construct a bad thing M for T as follows. Since $T \notin$ ***PNI***, there exists a trace t_L of low events that is produced by initial states s_1 and s_2 with differing probabilities. Let M be the prefix of T that completely determines the probability of t_L for those initial states:

$$M \;=\; \{t \in \Psi_{\mathsf{fin}} \mid t[0] \in \{s_1, s_2\} \,\wedge\, t \leq T \,\wedge\, ev_L(t) = t_L\}.$$

Recall that bad things must be finitely observable and irremediable. M is irremediable because no extension of it can change the probability of t_L for initial states s_1 and s_2. But is M finitely observable—that is, is $M \in \mathsf{Obs}$? Recall that an element of Obs must be a finite set of finite traces. Each trace in M is finite, but M might not be a finite set:

- If state space Σ is countably infinite,[10] there could be infinitely many states to which s_1 (and s_2) transition. Hence there could need to be infinitely many traces in M to completely determine the probability of t_L, so M could not be in Obs. Moreover, any finite subset N of M would necessarily omit some states from Σ. So it might be possible to extend N to a system T' that satisfies ***PNI*** by adding traces containing those omitted states. Thus T would have no bad thing, and ***PNI*** would not be hypersafety for ***PR***.
- If Σ is finite, only finitely many finite traces are low-equivalent to t_L. Thus M is finite, and no extension of T' of M can change the probability of t_L. So T' cannot be in ***PNI***. Therefore ***PNI*** is hypersafety for ***PR***.

Gray's definition of probabilistic noninterference [49] is hypersafety for ***PR***, because Gray required the state (and input and output) space to be finite. But the definition of O'Neill et al. [96] is neither hypersafety nor hyperliveness, because it allowed a countably infinite state space.

[10]State space Σ cannot be uncountably infinite without generalizing probability function $p(\cdot,\cdot)$ to a probability measure.

Secure encryption. A *private-key encryption scheme* is a tuple $(\mathcal{M}, \mathcal{K}, \mathcal{C}, \mathit{Gen}, \mathit{Enc}, \mathit{Dec})$, where $\mathcal{M}$ is the *message space*, $\mathcal{K}$ is the *key space*, and $\mathcal{C}$ is the *ciphertext space* such that the following hold:

- Gen is the *key-generation algorithm*, a randomized algorithm that produces a key $k \in \mathcal{K}$. We write $k \leftarrow \mathit{Gen}$ to denote the sampling of k from the probability distribution induced by Gen.
- Enc is the *encryption algorithm*, an algorithm (either randomized or deterministic) that accepts a key $k \in \mathcal{K}$, a plaintext message $m \in \mathcal{M}$, and yields a ciphertext $c \in \mathcal{C}$ that is the encryption of m using k. We denote this as $c = \mathit{Enc}(m, k)$.
- Dec is the *decryption algorithm*, a deterministic algorithm that accepts a key $k \in \mathcal{K}$, a ciphertext $c \in \mathcal{C}$, and yields a plaintext m that is the decryption of c using k. We denote this as $m = \mathit{Dec}(c, k)$.
- Decryption is the inverse of encryption. Formally, for all $m \in \mathcal{M}$ and $k \in \mathcal{K}$, it holds that $\Pr\left(\mathit{Dec}(\mathit{Enc}(m, k), k) = m\right) \;=\; 1$.

A private-key encryption scheme satisfies *perfect indistinguishability* [61] if the probability distribution on ciphertexts is the same for all plaintexts. Formally, for all m_1, m_2, and c,

$$\Pr\left(k \leftarrow \mathit{Gen} : \mathit{Enc}(m_1, k) = c\right) \;=\; \Pr\left(k \leftarrow \mathit{Gen} : \mathit{Enc}(m_2, k) = c\right).$$

Perfect indistinguishability can be formulated as a hyperproperty on probabilistic systems. To encode encryption scheme $(\mathcal{M}, \mathcal{K}, \mathcal{C}, \mathit{Gen}, \mathit{Enc}, \mathit{Dec})$ as a probabilistic system, let the set of states of the system be

$$\mathcal{M} \cup \mathcal{K} \cup \mathcal{C} \cup \{\mathit{Gen}\} \cup \{\mathit{Enc}(m, k) \mid k \in \mathcal{K}, m \in \mathcal{M}\} \\ \cup \{\mathit{Dec}(c, k) \mid k \in \mathcal{K}, c \in \mathcal{C}\}.$$

Let probability function $p(\cdot, \cdot)$ be defined such that

- $p(\mathit{Gen}, k) \;=\; \Pr(k = \mathit{Gen})$,
- $p(\mathit{Enc}(m, k), c) \;=\; \Pr(c = \mathit{Enc}(m, k))$, and
- $p(\mathit{Dec}(c, k), m) = 1$ iff $\mathit{Dec}(c, k) = m$.

Let the system so constructed from $(\mathcal{M}, \mathcal{K}, \mathcal{C}, \mathit{Gen}, \mathit{Enc}, \mathit{Dec})$ be denoted

$$\mathit{encSys}(\mathcal{M}, \mathcal{K}, \mathcal{C}, \mathit{Gen}, \mathit{Enc}, \mathit{Dec}),$$

and let the set of all such systems be ***ES***. The following hyperproperty expresses perfect indistinguishability:

$$\begin{aligned}
\boldsymbol{PI} \;\triangleq\; \{T \in \mathsf{Prop} \mid T \in \boldsymbol{ES} \\
&\wedge\; (\exists\, \mathcal{M}, \mathcal{K}, \mathcal{C}, \mathit{Gen}, \mathit{Enc}, \mathit{Dec} \;: \\
&\quad T = \mathit{encSys}(\mathcal{M}, \mathcal{K}, \mathcal{C}, \mathit{Gen}, \mathit{Enc}, \mathit{Dec}) \\
&\quad\quad \wedge\; (\forall\, m_1, m_2 \in \mathcal{M}; c \in \mathcal{C} \;: \\
&\quad\quad\quad \Pr(\mathit{Enc}(m_1) = c) \\
&\quad\quad\quad\quad = \; \Pr(\mathit{Enc}(m_2) = c)))\}, \qquad (5.5.2)
\end{aligned}$$

where $\Pr(\mathit{Enc}(m) = c)$ denotes

$$\sum_{k \in \mathcal{K}} \mathsf{Pr}_{\mathit{Gen},T}(\{\mathit{Gen}, k\}) \cdot \mathsf{Pr}_{\mathit{Enc}(m,k),T}(\{\mathit{Enc}(m, k), c\}).$$

PI is hypersafety for ***ES*** because any encryption scheme that is not in ***PI*** has a ciphertext c and two messages m_1, m_2 such that the probability that m_1 encrypts to c is not equal to the probability that m_2 encrypts to c. Trace set $\{\mathit{Enc}(m, k), c \mid k \in \mathcal{K}, m \in \{m_1, m_2\}\}$ thus is irremediable, and it is finite assuming that key space $\mathcal{K}$ is finite. So the trace set is a bad thing. But note that ***PI*** is not subset closed for Prop, so stepwise refinement is not applicable with ***PI***.

Other definitions of secure encryption, such as computational indistinguishability in various attacker models (including IND-CPA and IND-CCA), can similarly be formulated as hyperproperties.

Quantification of information flow. Probability can also be used to reason about the amount of information that a system can leak. For example, *channel capacity* is the maximum rate at which information can be reliably sent over a channel [106]; Gray [49] formulates as a channel the leakage of secret information from a system, and he quantifies the capacity of that channel. The hyperproperty "The channel capacity is k bits" (denoted $\boldsymbol{CC}_k$) is hyperliveness for $\boldsymbol{PR}$, since no matter what the rate is for some finite prefix of the system, the rate can changed to any arbitrary amount by an appropriate extension that conveys more or less information.

Chapter 2 gives a model and metric for quantifying the leakage over a series of experiments on a program S. The policy specifying that the leakage is less than k bits for all experiments, denoted $\boldsymbol{QL}_k$, is hypersafety for a variant of $\boldsymbol{PR}$, as we now show.

Recall that a *state* of a probabilistic program has an immutable high projection and a mutable low projection, that a *repeated experiment* on probabilistic program S is a finite sequence of executions of S, and that each individual execution is an *experiment*. An experiment can be represented with two states: an initial state, in which inputs are provided to the program, and a final state, in which outputs are given by the program. All initial states (across all executions) in a repeated experiment must have the same high projection but may have different low projections. Recall that the probabilistic behavior of S is modeled by a semantics $[\![S]\!]$ that maps inputs states to output distributions, where $([\![S]\!]s)(s')$

is the probability that S begun in state s terminates in state s'. An attacker begins an experiment with a *prebelief* about the high projection of the initial state. After observing the output of the execution, the attacker updates his prebelief to produce a *postbelief* about the high projection of the initial state.

We here use traces and events to represent repeated experiments, where each state in a trace produces an event. The events alternate between input and output, and the first event in a trace must be an input. Each output must have the correct probability of occurring according to $[\![S]\!]$ and the most recent input.[11] Each low input projection may vary, but the high projection must be the same in each input. Let $Syst(S)$ denote the system of such traces resulting from program S:

$$\begin{aligned} Syst(S) \;\triangleq\; \{t \in \Psi_{\mathsf{fin}} \mid (\forall\, i : 0 \le 2i+1 \le |t| \\ \implies ev_{Hin}(t[2i]) = ev_{Hin}(t[0]) \\ \wedge\; p(t[2i], t[2i+1]) = ([\![S]\!]t[2i])(t[2i+1]))\}, \end{aligned}$$

where $|t|$ denotes the length of finite trace t, and $p(\cdot,\cdot)$ is the probability function used in §5.5. From $Syst(S)$ we can construct probability measure $\mathsf{Pr}_{s,Syst(S)}$, also used in §5.5.[12]

Each pair of states $t[i]$ and $t[i+1]$, for even i, in repeated experiment t yields an experiment. An experiment is described formally by a prebelief, a high input, a low input, a low output, and a postbelief.

[11] A representation in which each finite trace contains two states (initial and final) might at first seem suitable for repeated experiments. That representation would fail to preserve the order in which inputs are provided (in initial states) across the sequence of executions in the repeated experiment. However, a single trace with many states does capture this order.

[12] Note that $p(s, s')$ is defined only at every other state in each trace of $Syst(S)$, so to construct the measure we treat each pair of states in the trace a single state. Also note that the set of program states must be finite for the probability measure to be well-defined.

As part of determining the postbelief for an experiment, the attacker's *prediction* δ_A of the low output is calculated from prebelief b_H and low input l:

$$\delta_A(b_H, l) \triangleq \lambda s \,.\, b_H(ev_{Hin}(s)) \cdot \mathsf{Pr}_{r,Syst(S)}(\{rs\}),$$

where r is the state that has $ev_{Hin}(s)$ as its high projection and l as its low projection. Denote the i^{th} experiment in trace t, with initial prebelief b_H, as $\mathcal{E}(t, i, b_H)$. We define $\mathcal{E}(t, i, b_H)$ using OCaml-style record syntax:

$$\begin{aligned}
\mathcal{E}(t, i, b_H) \triangleq \{ & \; preBelief = \text{if } i > 0 \text{ then } \mathcal{E}(t, i-1).postBelief \text{ else } b_H; \\
& highIn = ev_{Hin}(t[2i]); \\
& lowIn = ev_L(t[2i]); \\
& lowOut = ev_L(t[2i+1]); \\
& postBelief = (\delta_A(b_H, l) \mid lowOut) \upharpoonright H \; \},
\end{aligned}$$

where $\mid$ is the distribution conditioning operator, and $\upharpoonright$ is the distribution projection operator, defined in §2.1.

The quantity of flow in experiment $\mathcal{E}(t, i, b_H)$ is denoted $\mathcal{Q}(\mathcal{E}(t, i, b_H))$ and defined in §2.3.1. The quantity of flow over repeated experiment t with initial prebelief b_H, denoted $\mathcal{Q}(t, b_H)$, is the sum of the flow for each experiment in t:

$$\mathcal{Q}(t, b_H) \triangleq \sum_{i=0}^{(|t|-1)/2} \mathcal{Q}(\mathcal{E}(t, i, b_H)).$$

Hyperproperty $\boldsymbol{QL}_k$ is the set of all systems that exhibit at most k bits of flow over any experiment:

$$\boldsymbol{QL}_k \triangleq \{T \in \mathsf{Prop} \mid (\exists\, S : T = Syst(S) \implies (\forall\, t \in T, b_H : \mathcal{Q}(b_H, t) \leq k))\}.$$

5.6 Results on Generalized Hypersafety and Hyperliveness

The results proved in chapter 4 about hypersafety and hyperliveness generalize naturally to specific system representations.[13] Informally, the generalizations are as follows:

- If P is safety (liveness) for Rep, then $[P]$ is hypersafety (hyperliveness) for Rep (generalizing propositions 4.1 and 4.2).
- If $\boldsymbol{P}$ is hypersafety for Rep, then $\boldsymbol{P}$ is subset closed for Rep, but not necessarily subset closed for Prop (generalizing theorem 4.1). Consequently, stepwise refinement does not necessarily work with hyperproperties that are hypersafety for Rep.
- If $\boldsymbol{P}$ is a possibilistic information-flow policy for Rep, then $\boldsymbol{P}$ is hyperliveness for Rep (generalizing theorem 4.3).
- k-hypersafety for Rep can be reduced to safety for Rep^k (generalizing theorem 4.2).
- Every hyperproperty for Rep is the intersection of a safety hyperproperty for Rep with a liveness hyperproperty for Rep (generalizing theorem 4.5).

We give the formal statements of these generalized results below. The proofs of these results are all straightforward corollaries of the original results, although some proofs require additional assumptions about Rep.

First, we must define safety and liveness for system representations. Let $\mathit{Tr}(\mathsf{Rep})$ denote the set of all traces that are contained in any system in Rep—that is, $\mathit{Tr}(\mathsf{Rep}) = \bigcup_{T \in \mathsf{Rep}} T$. Let $\mathit{Obs}(\mathit{Tr}(\mathsf{Rep}))$ denote the set of all finite traces that are prefixes of some trace in $\mathit{Tr}(\mathsf{Rep})$—that is, $\mathit{Obs}(\mathit{Tr}(\mathsf{Rep})) = \{t \in \Psi_{\mathsf{fin}} \mid (\exists\, t' \in$

[13]We do not generalize the topological results here. However, since the intersection theorem generalizes, we believe that the topological results could also be generalized.

$\mathit{Tr}(\mathsf{Rep}) : t \le t')\}$. Let the lift $[P]_{\mathsf{Rep}}$ of property P in Rep be $\mathcal{P}(P) \cap \mathsf{Rep}$. A trace property S is a *safety property for system representation* Rep iff

$$(\forall\, t \in \mathit{Tr}(\mathsf{Rep}) : t \notin S \implies (\exists\, m \in \mathit{Obs}(\mathit{Tr}(\mathsf{Rep})) : m \le t \wedge (\forall\, t' \in \mathit{Tr}(\mathsf{Rep}) : m \le t' \implies t' \notin S))).$$

A trace property L is a *liveness property for system representation* Rep iff

$$(\forall\, t \in \mathit{Obs}(\mathit{Tr}(\mathsf{Rep})) : (\exists\, t' \in \mathit{Tr}(\mathsf{Rep}) : t \le t' \ \wedge\ t' \in L)).$$

Note that, compared to the original definitions of safety and liveness in chapter 4, we have simply replaced Ψ_{inf} with $\mathit{Tr}(\mathsf{Rep})$, and Ψ_{fin} with $\mathit{Obs}(\mathit{Tr}(\mathsf{Rep}))$. Let $\mathsf{SP}(\mathsf{Rep})$ be the set of all safety properties for Rep, and let $\mathsf{LP}(\mathsf{Rep})$ be the set of all liveness properties for Rep. Likewise, let $\mathbf{SHP}(\mathsf{Rep})$ be the set of all safety hyperproperties for Rep, and let $\mathbf{LHP}(\mathsf{Rep})$ be the set of all liveness hyperproperties for Rep.

Generalization of proposition 4.1. If $(\forall\, t \in \mathit{Tr}(\mathsf{Rep}) : \{t\} \in \mathsf{Rep})$, then

$$(\forall\, S \in \mathcal{P}(\mathsf{Rep}) : S \in \mathsf{SP}(\mathsf{Rep}) \iff [S]_{\mathsf{Rep}} \in \mathbf{SHP}(\mathsf{Rep})).$$

The forward direction of this generalization always holds, but the backward direction ($\Longleftarrow$) might not hold if Rep does not allow individual traces from $\mathit{Tr}(\mathsf{Rep})$ to be representations: the bad thing for a safety hyperproperty could never be an individual trace, hence the safety hyperproperty could not be the lift of a safety property. So the backward direction requires the assumption that any individual trace in $\mathit{Tr}(\mathsf{Rep})$ is itself a system representation in Rep—that is, $(\forall\, t \in \mathit{Tr}(\mathsf{Rep}) : \{t\} \in \mathsf{Rep})$. Note that Prop satisfies this assumption.

Generalization of proposition 4.2. If $(\forall\, T \subseteq \mathit{Tr}(\mathsf{Rep}) : T \in \mathsf{Rep})$, then

$$(\forall\, L \in \mathcal{P}(\mathsf{Rep}) : L \in \mathsf{LP}(\mathsf{Rep}) \iff [L]_{\mathsf{Rep}} \in \mathbf{LHP}(\mathsf{Rep})).$$

The backward direction of this generalization always holds, but the forward direction ($\Longrightarrow$) might not hold if Rep does not allow arbitrary unions of individual traces from $\mathit{Tr}(\mathsf{Rep})$ to be representations: the union of the individual good things for a liveness property would not necessarily be good for the lift of that liveness property. So the forward direction requires the assumption that arbitrary unions of individual traces in $\mathit{Tr}(\mathsf{Rep})$ are themselves system representations in Rep—that is, $(\forall\, T \subseteq \mathit{Tr}(\mathsf{Rep}) \,:\, T \in \mathsf{Rep})$. Note that Prop satisfies this assumption.

Generalization of theorem 4.1. If $(\exists\, L \in \mathsf{LP}(\mathsf{Rep}) \,:\, L \neq \mathit{Tr}(\mathsf{Rep}))$, then

$$\mathbf{SHP}(\mathsf{Rep}) \subset \mathbf{SSC}(\mathsf{Rep}).$$

$\mathsf{SSC}(\mathsf{Rep})$ is the set of all hyperproperties for Rep that are subset closed on Rep:

$$\boldsymbol{P} \in \mathbf{SSC}(\mathsf{Rep}) \iff (\forall\, T \in \boldsymbol{P} \,:\, (\forall\, T' \in \mathsf{Rep} \,:\, T' \subset T \implies T' \in \boldsymbol{P})).$$

The strictness of the subset in the theorem generalization requires the assumption that there exist subset-closed hyperproperties that are not safety. But it suffices to instead assume that hyperliveness is not trivial for Rep—that is, $(\exists\, L \in \mathsf{LP}(\mathsf{Rep}) \,:\, L \neq \mathit{Tr}(\mathsf{Rep}))$. Note that Prop satisfies both assumptions.

Generalization of theorem 4.2.

$$(\forall\, S \in \mathsf{Rep}, \boldsymbol{K} \in \mathbf{KSHP(}k\mathbf{)}(\mathsf{Rep}) \,:\, (\exists\, K \in \mathsf{SP}(\mathsf{Rep}) \,:\, S \models \boldsymbol{K} \iff S^k \models K)).$$

$\mathbf{KSHP(}k\mathbf{)}(\mathsf{Rep})$ is the subset of $\mathbf{SHP}(\mathsf{Rep})$ where the size of bad thing M is bounded by k.

Generalization of theorem 4.3. If there exists some liveness hyperproperty for Rep that is not a possibilistic information-flow policy for Rep, then

$$\mathbf{PIF}(\mathsf{Rep}) \subset \mathbf{LHP}(\mathsf{Rep}).$$

$\mathbf{PIF}(\mathsf{Rep})$ is the set of all possibilistic information-flow policies expressed by closure operators Cl of type $\mathsf{Rep} \to \mathsf{Rep}$. The strictness of the subset requires the assumption of the existence of a liveness hyperproperty for Rep that is not a possibilistic information-flow policy for Rep. Note that Prop satisfies this assumption.

Generalization of theorem 4.5.

$$(\forall\, \boldsymbol{P} \in \mathcal{P}(\mathsf{Rep}) : (\exists\, \boldsymbol{S} \in \mathbf{SHP}(\mathsf{Rep}), \boldsymbol{L} \in \mathbf{LHP}(\mathsf{Rep}) : \boldsymbol{P} = \boldsymbol{S} \cap \boldsymbol{L})).$$

The proof of this generalization requires the following generalized definition:

$$\begin{aligned} \mathit{Safe}(\boldsymbol{P}) \;\triangleq\; \{T \in \mathsf{Rep} \mid (\forall\, M \in \mathit{Obs}(\mathsf{Rep}) : M \leq T \\ \implies (\exists\, T' \in \mathsf{Rep} : M \leq T' \;\wedge\; T' \in \boldsymbol{P}))\}. \end{aligned}$$

Also, in the definition of $\mathit{Live}(\boldsymbol{P})$, notation $\overline{\boldsymbol{H}}$ must now denote the complement of hyperproperty $\boldsymbol{H}$ with respect to Rep.

5.7 Summary

This chapter has classified several security policies with hypersafety and hyperliveness for particular system representations. Figure 5.1 summarizes this classification.

We have shown that the theory of hyperproperties can be generalized to apply to system representations such as relational semantics, labeled transition

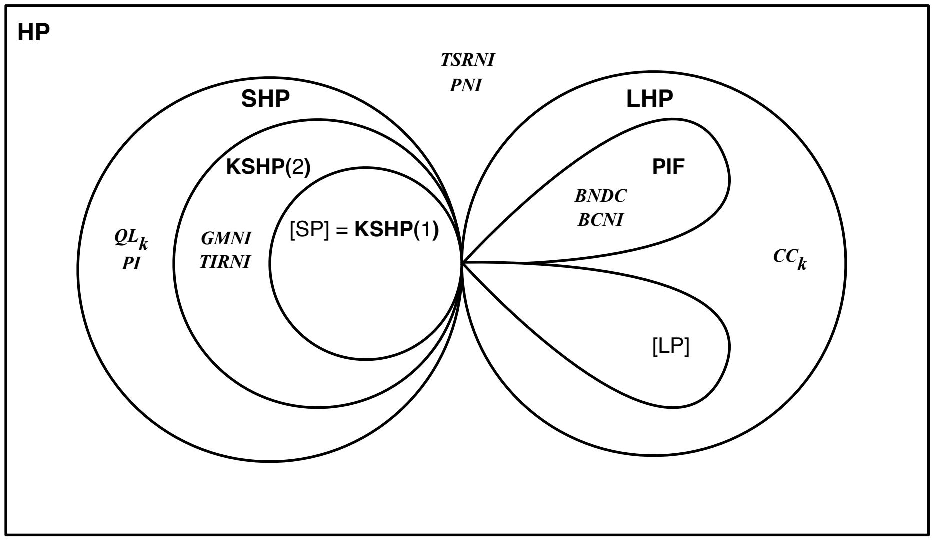

Figure 5.1: Classification of security policies for system representations

systems, state machines, and probabilistic systems. In each case, we encode the system representation into trace sets, thus into hyperproperties. All of our theorems about hyperproperties continue to hold for system representations, though some additional assumptions about the system representation are needed.

CHAPTER 6

CONCLUSION

In practice, computer security policies are often expressed as informal requirements in natural languages (e.g., English), which are inherently ambiguous. But security policies can also be expressed precisely with mathematical models and notations, and this precision makes policies amenable to analysis both by humans and computers.

This dissertation has developed such mathematical foundations. Information theory was used in chapters 2 and 3 to quantify information-flow security. This quantification is useful for analyzing the security of systems whose proper operation requires leakage of information, such as password checkers and statistical databases. We showed that accuracy of belief can be used to quantify information flow for both confidentiality and integrity, and that accuracy generalizes previous metrics based on uncertainty. Hyperproperties were used in chapters 4 and 5 to formalize security policies. This formalization is the first to enable expression of all kinds of security requirements in a uniform framework. We showed that the theory of trace properties generalizes to hyperproperties.

The historical background in §1.1 began with the taxonomy of confidentiality, integrity, and availability. More research is needed on the relationship between this taxonomy and the formalisms we have studied. For quantitative flow, we have given definitions for confidentiality and integrity, but availability remains unexplored. For hyperproperties, the relationship with the taxonomy is an open question, but we can offer some observations:

- Information-flow confidentiality is not a trace property, but it is a hyperproperty, and it can be hypersafety (e.g., observational determinism) or hyperliveness (e.g., generalized noninterference).

- Integrity, as the information-flow dual of confidentiality, includes examples from both hypersafety and hyperliveness. And when stipulating access control on changes to data and other resources, integrity is safety.
- Availability is sometimes hypersafety (maximum response time in any execution, which is also safety) and sometimes hyperliveness (mean response time over all executions).

The classification of security requirements as confidentiality, integrity, and availability therefore would seem to be orthogonal to hypersafety and hyperliveness.

More research is also needed on how to obtain assurance that real systems meet the security definitions we have given. For quantitative flow, one important open question is how to make our theoretical policies practical in real systems, either by enforcing a limit on information flow or by measuring the actual amount of information flow. For hyperproperties, we gave a relatively complete verification methodology for k-hypersafety properties, but whether there is a relatively complete verification methodology for all hyperproperties remains an important open question.

The immediate goal of the research presented in this dissertation is to improve our understanding of the foundations of computer security so that we can specify system security requirements and gain assurance that systems meet those requirements. But the ultimate goal is to ameliorate the real-world consequences of security vulnerabilities. These vulnerabilities were a motivation for the 1991 report by the System Security Study Committee of the National Research Council:

> "Computer systems are coming of age. As [they] become more prevalent, sophisticated, ...and interconnected, society becomes

more vulnerable to poor system design, accidents..., and attacks. Without more responsible design and use, system disruptions will increase, with harmful consequences for society." [92, Executive Summary]

Now, almost two decades later, it seems clear not only that the Committee was right, but that the potential for disruptions and the severity of their consequences continues to increase. It is my hope that the research presented in this dissertation will in some way help to reduce the economic, defense, and social consequences of security vulnerabilities.

BIBLIOGRAPHY

[1] Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, May 1991.

[2] Martín Abadi and Leslie Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 15(1):73–132, 1993.

[3] Samson Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 51:1–77, 1991.

[4] Jiří Adámek. *Foundations of Coding*. John Wiley and Sons, New York, 1991.

[5] Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, 1985.

[6] Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2(3):117–126, 1987.

[7] James P. Anderson. Computer security technology planning study. Technical Report ESD-TR-73-51, Vol. II, Electronic Systems Division, Air Force Systems Command, Bedford, Mass., October 1972.

[8] Ross J. Anderson. A security policy model for clinical information systems. In *Proc. IEEE Symposium on Security and Privacy*, pages 30–43, May 1996.

[9] Ralph-Johan R. Back. On correct refinement of programs. *Journal of Computer and System Sciences*, 23(1):49–68, August 1981.

[10] Michael Backes. Quantifying probabilistic information flow in computational reactive systems. In *Proc. European Symposium on Research in Computer Security*, pages 336–354, September 2005.

[11] Michael Backes, Boris Köpf, and Andrey Rybalchenko. Automated discovery and quantification of information leaks. In *Proc. IEEE Symposium on Security and Privacy*, pages 141–153, May 2009.

[12] Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. Secure information flow by self-composition. In *Proc. IEEE Computer Security Foundations Workshop*, pages 100–114, June 2004.

[13] D. Elliot Bell and Leonard J. LaPadula. Secure computer systems: Mathematical foundations. Technical Report 2547, Volume I, MITRE Corporation, March 1973.

[14] Johan van Benthem and Kees Doets. Higher-order logic. In *Elements of Classical Logic*, volume 1 of *Handbook of Philosophical Logic*. D. Reidel Publishing, Dordrecht, Holland, 1983.

[15] Kenneth Biba. Integrity considerations for secure computer systems. Technical Report MTR-3153, MITRE Corporation, April 1977.

[16] Matt Bishop. *Computer Security: Art and Science*. Addison-Wesley, Boston, 2003.

[17] Annalisa Bossi, Riccardo Focardi, Carla Piazza, and Sabina Rossi. Refinement operators and information flow security. In *IEEE Conference on Software Engineering and Formal Methods*, pages 44–53, June 2003.

[18] Gérard Boudol and Ilaria Castellani. Noninterference for concurrent programs and thread systems. *Theoretical Computer Science*, 281(1–2):109–130, 2002.

[19] Randy Browne. The Turing test and non-information flow. In *Proc. IEEE Symposium on Security and Privacy*, pages 375–385, May 1991.

[20] Denis L. Bueno and Michael R. Clarkson. Hyperproperties: Verification of proofs. Technical report, Cornell University Computing and Information Science, July 2008. Available from `http://hdl.handle.net/1813/11153`.

[21] Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Prakash Panangaden. Anonymity protocols as noisy channels. *Information and Computation*, 206(2–4):378–401, 2008.

[22] Stephen Chong and Andrew C. Myers. Decentralized robustness. In *Proc. IEEE Computer Security Foundations Workshop*, pages 242–253, July 2006.

[23] David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantified interference: Information theory and information flow. In *IFIP WG 1.7 Workshop on Issues in the Theory of Security*, Barcelona, Spain, April 2004. Available from King's College London Computer Science E-Repository, document ID 1107, `http://calcium.dcs.kcl.ac.uk/1107`.

[24] David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantified interference for a while language. *Electronic Notes in Theoretical Computer Science*, 112:149–166, January 2005.

[25] David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantitative information flow, relations and polymorphic types. *Journal of Logic and Computation*, 18(2):181–199, 2005.

[26] David D. Clark and David R. Wilson. A comparison of commercial and military computer security policies. In *Proc. IEEE Symposium on Security and Privacy*, pages 184–194, April 1987.

[27] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[28] Michael R. Clarkson, Andrew C. Myers, and Fred B. Schneider. Belief in information flow. In *Proc. IEEE Computer Security Foundations Workshop*, pages 31–45, June 2005.

[29] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. In *Proc. IEEE Symposium on Computer Security Foundations*, pages 51–65, June 2008.

[30] Commission of the European Communities (ECSC, EEC, EAEC). Information Technology Security Evaluation Criteria: Provisional harmonised criteria, June 1991. Document COM(90) 314, Version 1.2.

[31] Common Criteria for Information Technology Security Evaluation: Part 1: Introduction and general model, September 2005. CCMB-2006-09-001, Version 3.1, Revision 1. Available from `www.commoncriteriaportal.org`.

[32] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley & Sons, New York, 1991.

[33] Ole-Johan Dahl, C.A.R. Hoare, and Edsger W. Dijkstra. *Structured Programming*. Academic Press, London, 1972.

[34] Ádám Darvas, Reiner Hähnle, and David Sands. A theorem proving approach to analysis of secure information flow. In *Security in Pervasive Computing*, volume 3450 of *Lecture Notes in Computer Science*, pages 193–209. Springer, Berlin, 2005.

[35] Dorothy Denning. *Cryptography and Data Security*. Addison-Wesley, Reading, Massachusetts, 1982.

[36] Dorothy E. Denning. A lattice model of secure information flow. *Communications of the ACM*, pages 236–242, May 1976.

[37] Department of Defense. Trusted Computer System Evaluation Criteria, December 1985. DoD 5200.28-STD, also known as the "Orange Book".

[38] Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Approximate non-interference. *Journal of Computer Security*, 12(1):37–81, 2004.

[39] Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Measuring the confinement of probabilistic systems. *Theoretical Computer Science*, 340(1):3–56, 2005.

[40] Edsger W. Dijkstra. A constructive approach to the problem of program correctness. *BIT Numerical Mathematics*, 8:174–186, 1968.

[41] E. Allen Emerson and Joseph Y. Halpern. "Sometimes" and "not never" revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, January 1986.

[42] Alexandre Evfimievski, Johannes Gehrke, and Ramakrishnan Srikant. Limiting privacy breaches in privacy preserving data mining. In *Proc. ACM Symposium on Principles of Database Systems*, pages 211–222, June 2003.

[43] R. J. Feiertag, K. N. Levitt, and L. Robinson. Proving multilevel security of a system design. In *Proc. ACM Symposium on Operating Systems Principles*, pages 57–65, November 1977.

[44] Riccardo Focardi and Roberto Gorrieri. Classification of security properties (part I: Information flow). In Riccardo Focardi and Roberto Gorrieri, editors, *Proc. International School on Foundations of Security Analysis and Design*, volume 2171 of *Lecture Notes in Computer Science*, pages 331–396. Springer, 2001.

[45] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC, Boca Raton, Florida, second edition, 2004.

[46] Joseph A. Goguen and Jose Meseguer. Security policies and security models. In *Proc. IEEE Symposium on Security and Privacy*, pages 11–20, April 1982.

[47] Dieter Gollmann. *Computer Security*. John Wiley and Sons, Chichester, 1999.

[48] James W. Gray, III. Probabilistic interference. In *Proc. IEEE Symposium on Security and Privacy*, pages 170–179, May 1990.

[49] James W. Gray, III. Toward a mathematical foundation for information flow security. In *Proc. IEEE Symposium on Security and Privacy*, pages 21–35, May 1991.

[50] James W. Gray, III and Paul F. Syverson. A logical approach to multilevel security of probabilistic systems. *Distributed Computing*, 11(2):73–90, 1998.

[51] Joseph Halpern and Kevin O'Neill. Secrecy in multiagent systems. In *Proc. IEEE Computer Security Foundations Workshop*, pages 32–46, June 2002.

[52] Joseph Y. Halpern. *Reasoning about Uncertainty*. MIT Press, Cambridge, Massachusetts, 2003.

[53] Joseph Y. Halpern and Mark R. Tuttle. Knowledge, probability, and adversaries. *Journal of the ACM*, 40(4):917–962, 1993.

[54] Godfrey Harold Hardy. *Divergent Series*. Chelsea, New York, 1991.

[55] Jifeng He, C.A.R. Hoare, and Jeff W. Sanders. Data refinement refined. In *Proc. European Symposium on Programming*, pages 187–196, March 1986.

[56] C.A.R. Hoare. Proof of correctness of data representations. *Acta Informatica*, 1:271–281, 1972.

[57] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, Englewood Cliffs, New Jersey, 1985.

[58] International Organization for Standardization. Information processing systems: Open systems interconnection—basic reference model. Part 2: Security architecture, 1989. ISO 7498-2.

[59] Gareth A. Jones and J. Mary Jones. *Information and Coding Theory*. Springer, London, 2000.

[60] Daniel Kahneman and Amos Tversky. Subjective probability: A judgment of representativeness. *Cognitive Psychology*, 3:430–454, 1972.

[61] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC, Boca Raton, Florida, 2008.

[62] Richard A. Kemmerer. Shared resource matrix methodology: An approach to identifying storage and timing channels. *ACM Transactions on Computing Systems*, 1(3):256–277, August 1983.

[63] Daniel Kifer and Johannes Gehrke. Injecting utility into anonymized datasets. In *Proc. ACM Conference on Management of Data*, pages 217–228, June 2006.

[64] Johnathan J. Koehler. The base rate fallacy reconsidered: Descriptive, normative, and methodological challenges. *Behavioral and Brain Sciences*, 19(1):1–53, 1996.

[65] Boris Köpf and David Basin. An information-theoretic model for adaptive side-channel attacks. In *Proc. ACM Conference on Computer and Communications Security*, pages 286–296, October 2007.

[66] Dexter Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22:328–350, 1981.

[67] Maxwell Krohn, Alexander Yip, Micah Brodsky, Natan Cliffer, M. Frans Kaashoek, Eddie Kohler, and Robert Morris. Information flow control for standard OS abstractions. In *Proc. ACM Symposium on Operating Systems Principles*, pages 321–334, October 2007.

[68] Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, 3(2):125–143, 1977.

[69] Leslie Lamport. "Sometime" is sometimes "not never": On the temporal logic of programs. In *Proc. ACM Symposium on Principles of Programming Languages*, pages 174–185, January 1980.

[70] Leslie Lamport. Basic concepts: Logical foundation. In *Distributed Systems: Methods and Tools for Specification, An Advanced Course*, volume 190 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 1985.

[71] Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, Boston, 2002.

[72] Butler W. Lampson. Computer security in the real world. Presented at *Annual Computer Security Applications Conference*, 2000. Available from `http://research.microsoft.com/en-us/um/people/blampson/64-securityinrealworld/Acrobat.pdf`.

[73] Butler W. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, October 1973.

[74] Butler W. Lampson. Protection. *ACM SIGOPS Operating Systems Review*, 8(1):18–24, January 1974.

[75] V. Benjamin Livshits and Monica S. Lam. Finding security vulnerabilities in Java applications with static analysis. In *Proc. USENIX Security Symposium*, pages 271–286, August 2005.

[76] Gavin Lowe. Quantifying information flow. In *Proc. IEEE Computer Security Foundations Workshop*, pages 18–31, June 2002.

[77] Pasquale Malacaria. Assessing security threats of looping constructs. In *Proc. ACM Symposium on Principles of Programming Languages*, pages 225–235, January 2007.

[78] Heiko Mantel. Possibilistic definitions of security: An assembly kit. In *Proc. IEEE Computer Security Foundations Workshop*, pages 185–199, July 2000.

[79] Heiko Mantel. Preserving information flow properties under refinement. In *Proc. IEEE Symposium on Security and Privacy*, pages 78–91, May 2001.

[80] Stephen McCamant and Michael D. Ernst. Quantitative information flow as network capacity. In *Proc. ACM Conference on Programming Language Design and Implementation*, pages 193–205, June 2008.

[81] Daryl McCullough. Specifications for multi-level security and a hook-up property. In *Proc. IEEE Symposium on Security and Privacy*, pages 161–166, April 1987.

[82] Annabelle McIver and Carroll Morgan. A probabilistic approach to information hiding. In *Programming Methodology*, chapter 20, pages 441–460. Springer, New York, 2003.

[83] Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer, New York, 2004.

[84] John McLean. Security models and information flow. In *Proc. IEEE Symposium on Security and Privacy*, pages 180–189, May 1990.

[85] John McLean. Proving noninterference and functional correctness using traces. *Journal of Computer Security*, 1(1):37–58, 1992.

[86] John McLean. A general theory of composition for a class of "possibilistic" properties. *IEEE Transactions on Software Engineering*, 22(1):53–67, 1996.

[87] Ernest Michael. Topologies on spaces of subsets. *Transactions of the American Mathematical Society*, 71(1):152–182, July 1951.

[88] Jonathan Millen. Covert channel capacity. In *Proc. IEEE Symposium on Security and Privacy*, pages 60–66, April 1987.

[89] Jonathan Millen. 20 years of covert channel modeling and analysis. In *Proc. IEEE Symposium on Security and Privacy*, pages 113–114, May 1999.

[90] Robin Milner. *Communication and Concurrency*. Prentice Hall, New York, 1989.

[91] National Computer Security Center. A guide to understanding covert channel analysis of trusted systems. Technical Guideline NCSC-TG-030, Fort Meade, Maryland, November 1993.

[92] National Research Council. *Computers at Risk: Safe Computing in the Information Age*. National Academy Press, Washington, D.C., 1991.

[93] James Newsome and Dawn Song. Dynamic taint analysis for automatic detection, analysis and signature generation of exploits on commodity software. In *Proc. Symposium on Network and Distributed System Security*,

San Diego, California, February 2005. Available from http://www.isoc.org/isoc/conferences/ndss/05/proceedings/papers/taintcheck.pdf.

[94] James Newsome, Dawn Song, and Stephen McCamant. Measuring channel capacity to distinguish undue influence. In *Proc. ACM Workshop on Programming Languages and Analysis for Security*, Dublin, Ireland, June 2009. Available from http://doi.acm.org/10.1145/1554339.1554349.

[95] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Springer-Verlag, Berlin, 2002.

[96] Kevin R. O'Neill, Michael R. Clarkson, and Stephen Chong. Information-flow security for interactive programs. In *Proc. IEEE Computer Security Foundations Workshop*, pages 190–201, July 2006.

[97] Charles P. Pfleeger. *Security in Computing*. Prentice Hall PTR, Upper Saddle River, New Jersey, second edition, 1997.

[98] Gordon Plotkin. Domains. Available from http://homepages.inf.ed.ac.uk/gdp/publications/Domains.ps, 1983.

[99] Francois Pottier and Vincent Simonet. Information flow inference for ML. In *Proc. ACM Symposium on Principles of Programming Languages*, pages 319–330, January 2002.

[100] Riccardo Pucella and Fred B. Schneider. Independence from obfuscation: A semantic framework for diversity. In *Proc. IEEE Computer Security Foundations Workshop*, pages 230–241, July 2006.

[101] Lyle Harold Ramshaw. *Formalizing the Analysis of Algorithms*. PhD thesis, Stanford University, 1979. XEROX PARC technical report, 1981.

[102] A. W. Roscoe. CSP and determinism in security modelling. In *Proc. IEEE Symposium on Security and Privacy*, pages 114–127, May 1995.

[103] John Rushby. Security requirements specifications: How and what? (extended abstract). Invited paper presented at *Symposium on Requirements Engineering for Information Security*, Indianapolis, Indiana, March 2001. Available from http://www.csl.sri.com/users/rushby/abstracts/sreis01.

[104] Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, January 2003.

[105] Fred B. Schneider. *On Concurrent Programming*. Springer, New York, 1997.

[106] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.

[107] Stewart Shapiro. *Foundations without Foundationalism: A Case for Second-order Logic*. Clarendon Press, Oxford, 1991.

[108] Geoffrey Smith. On the foundations of quantitative information flow. In *Proc. Conference on Foundations of Software Science and Computation Structures*, pages 288–302, March 2009.

[109] Michael B. Smyth. Power domains and predicate transformers: A topological view. In *Proc. International Colloquium on Automata, Languages, and Programming*, pages 662–675, July 1983.

[110] Michael B. Smyth. Topology. In *Background: Mathematical Structures*, volume 1 of *Handbook of Logic in Computer Science*. Oxford University Press, 1992.

[111] Daniel F. Sterne. On the buzzword "security policy". In *Proc. IEEE Symposium on Security and Privacy*, pages 219–230, May 1991.

[112] G. Edward Suh, Jae W. Lee, David Zhang, and Srinivas Devedas. Secure program execution via dynamic information flow tracking. In *Proc. ACM Conference on Architectural Support for Programming Languages and Systems*, pages 85–96, October 2004.

[113] David Sutherland. A model of information. In *Proc. National Computer Security Conference*, pages 175–183, September 1986.

[114] Tad Taylor. Comparison paper between the Bell and LaPadula model and the SRI model. In *Proc. IEEE Symposium on Security and Privacy*, pages 195–202, April 1984.

[115] Tachio Terauchi and Alexander Aiken. Secure information flow as a safety problem. In *Proc. ACM Symposium on Static Analysis*, pages 352–367, September 2005.

[116] Leopold Vietoris. Bereiche zweiter Ordnung. *Monatschefte für Mathematik und Physik*, 33:49–62, 1923.

[117] Dennis Volpano. Safety versus secrecy. In *Proc. ACM Symposium on Static Analysis*, pages 303–311, September 1999.

[118] Dennis Volpano. Secure introduction of one-way functions. In *Proc. IEEE Computer Security Foundations Workshop*, pages 246–254, July 2000.

[119] Dennis Volpano and Geoffrey Smith. Confinement properties for programming languages. *SIGACT News*, 29(3):33–42, September 1998.

[120] Dennis Volpano and Geoffrey Smith. Verifying secrets and relative secrecy. In *Proc. ACM Symposium on Principles of Programming Languages*, pages 268–276, January 2000.

[121] Victor L. Voydock and Stephen T. Kent. Security mechanisms in high-level network protocols. *Computing Surveys*, 15(2), June 1983.

[122] Larry Wall, Tom Christiansen, and Randal L. Schwartz. *Programming Perl*. O'Reilly, Sebastopol, California, second edition, 1996.

[123] Douglas G. Weber. Quantitative hook-up security for covert channel analysis. In *Proc. IEEE Computer Security Foundations Workshop*, pages 58–71, June 1988.

[124] C. Weissman. Security controls in the ADEPT-50 time-sharing system. In *Proc. AFIPS Fall Joint Computer Conference*, pages 119–133, November 1969.

[125] Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, Cambridge, Massachusetts, 1993.

[126] Niklaus Wirth. Program development by stepwise refinement. *Communications of the ACM*, 14(4):221–227, April 1971.

[127] J. Todd Wittbold and Dale Johnson. Information flow in nondeterministic systems. In *Proc. IEEE Symposium on Security and Privacy*, pages 144–161, May 1990.

[128] Wei Xu, Sandeep Bhatkar, and R. Sekar. Taint-enhanced policy enforcement: A practical approach to defeat a wide range of attacks. In *Proc. USENIX Security Symposium*, pages 121–136, August 2006.

[129] Aris Zakinthinos and E.S. Lee. A general theory of security properties. In *Proc. IEEE Symposium on Security and Privacy*, pages 94–102, May 1997.

[130] Steve Zdancewic and Andrew C. Myers. Observational determinism for concurrent program security. In *Proc. IEEE Computer Security Foundations Workshop*, pages 29–43, June 2003.

[131] Lantian Zheng and Andrew C. Myers. End-to-end availability policies and noninterference. In *Proc. IEEE Computer Security Foundations Workshop*, pages 272–286, June 2005.

[132] Lidong Zhou, Fred B. Schneider, and Robbert van Renesse. APSS: Proactive secret sharing in asynchronous systems. *ACM Transactions on Information and System Security*, 8(3):259–286, August 2005.

INDEX